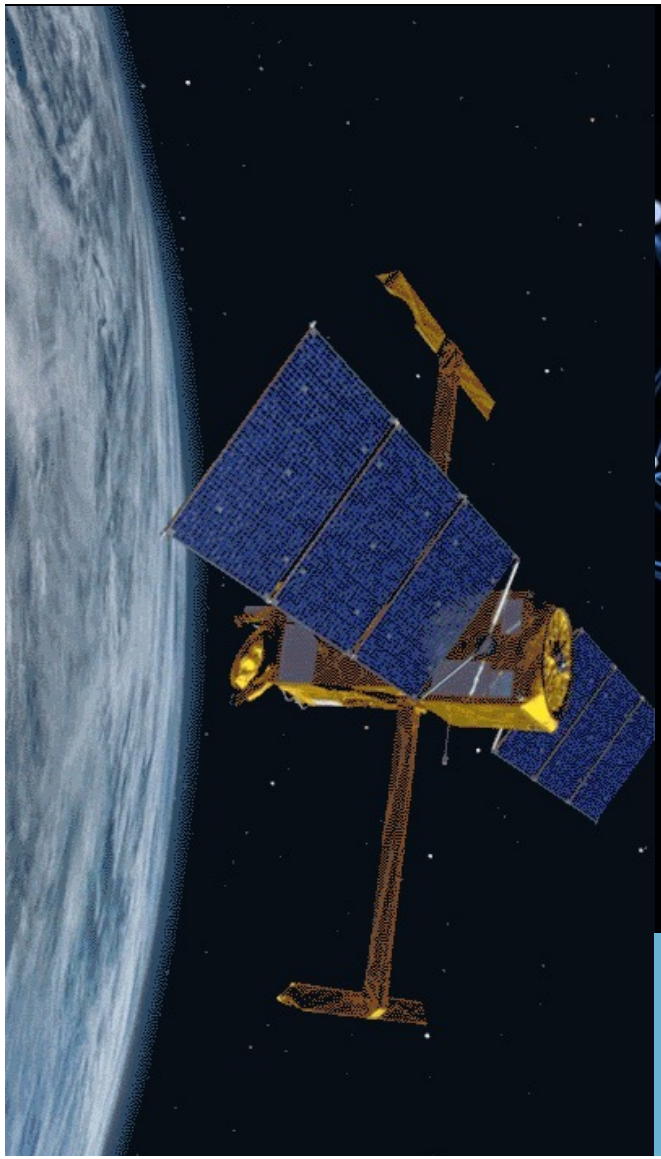


Lecture 5
Classification by Logistic Regression

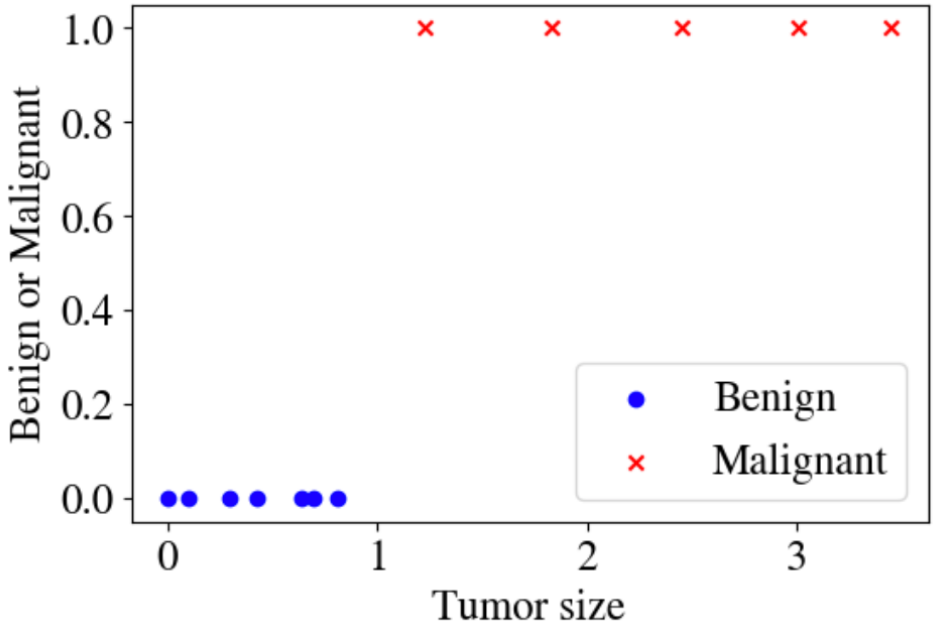
Problem statement

- Given one or more features, predict a *discrete* target *y*
- Target labels *y* for different **classes**, *y* = 0, 1, 2,…, N
- Examples:
    1. Spam filter, classes: regular email (*y*=0) or spam (*y*=1),
    2. Computer vision: picture is of cat (*y*=0), dog (*y*=1), llama (*y*=3), etc.,
    3. Computer vision: image pixels are water, agriculture, forest fire, etc.
    4. Weather: cloudy, rainy, sunny, stormy,
    5. Mineralogy: grain is feldspar, quartz, olivine, …
    6. Hydrology/glaciology: water is liquid, ice, vapor, snow.
- **Binary** or **multiclass classification**
- Features can be continuous/numerical or discrete/**categorical**

First, Univariate Binary Classification

Example

- Continuous feature *x* is tumor size,
- Classes are either benign ($y = 0$) or malignant ($y = 1$)

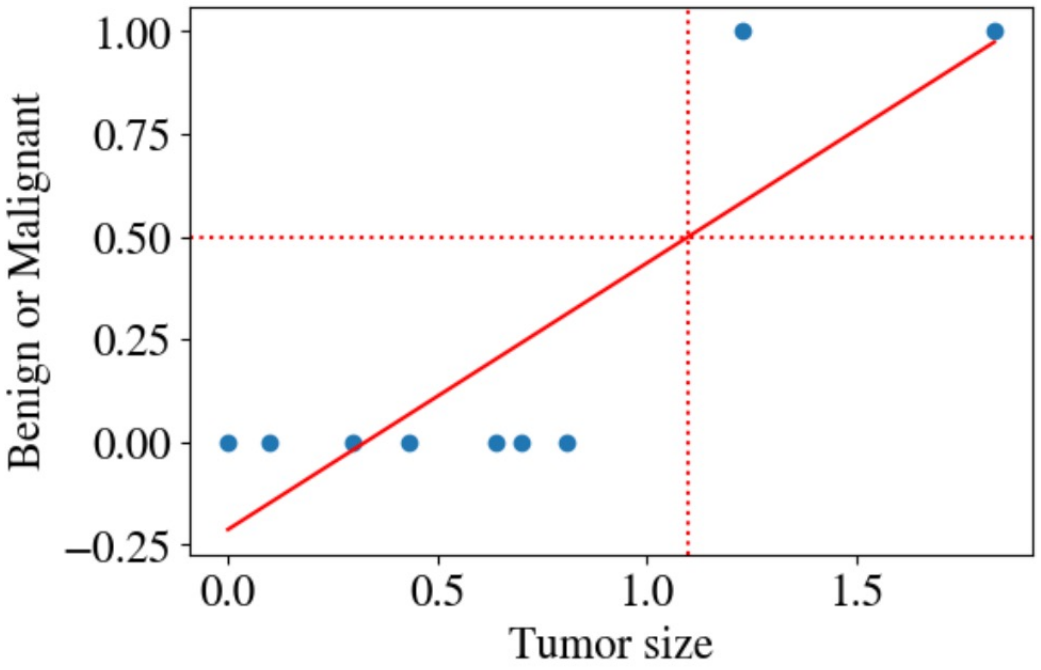


Numerical regression fit?

- Let's say we tried to predict benign/malignant with linear model:
- y = -0.21 + 0.65 x, R^2=0.72

- We could threshold as
  - Malignant for $y > 0.5$, $x = 1.1$,
  - Benign for $y < 0.5$, $x = 1.1$

- But…

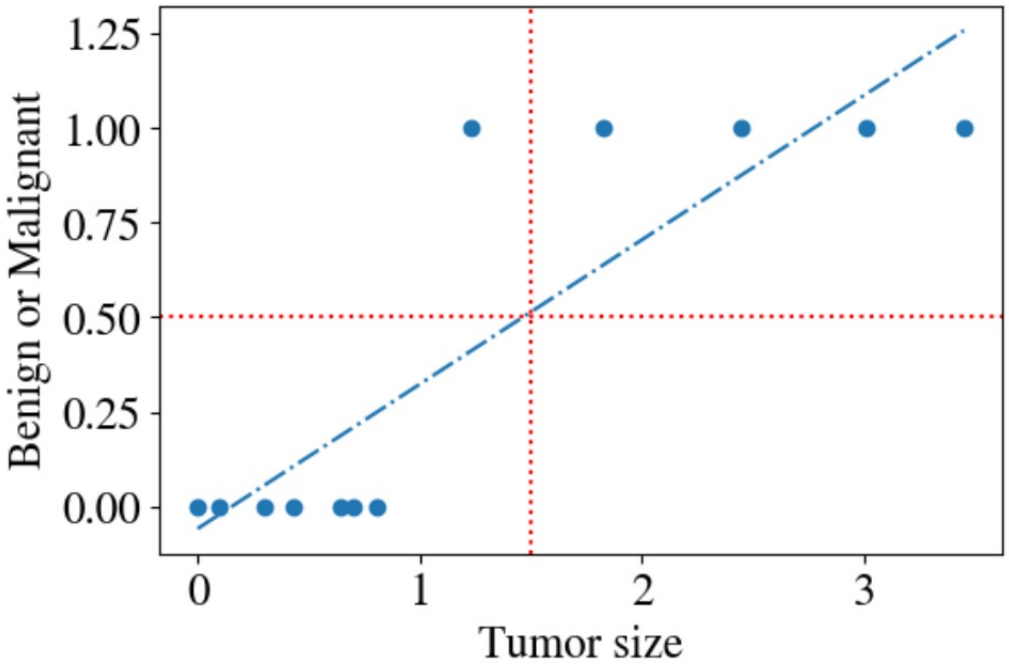


Numerical regression fit?

- What happens when we have few more measurements:
- Threshold of $y = 0.5$ doesn't work!

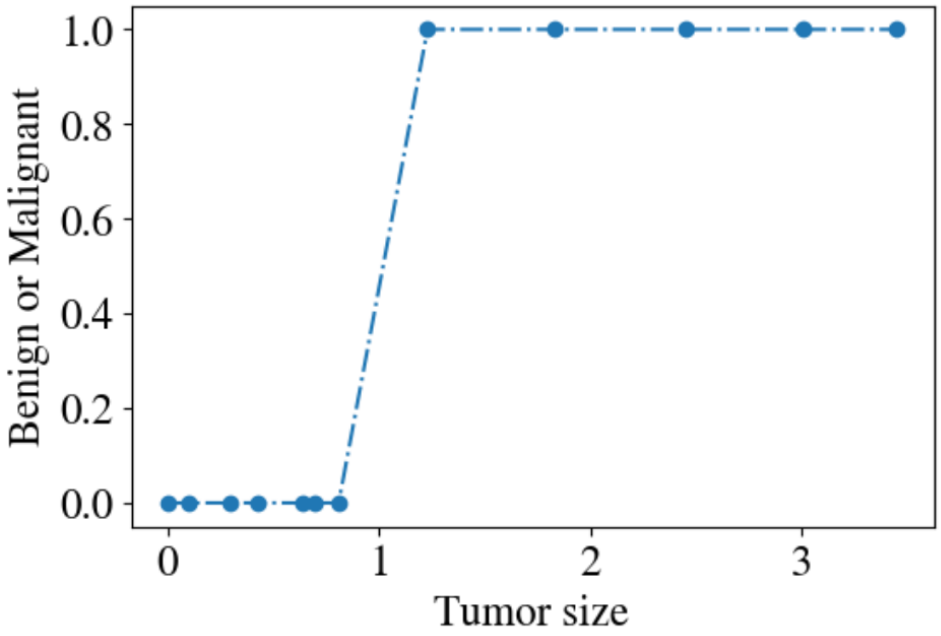
- Also, $y < 0$ and $y > 1$ for some x
- For any polynomial model

- Numerical regression no good.

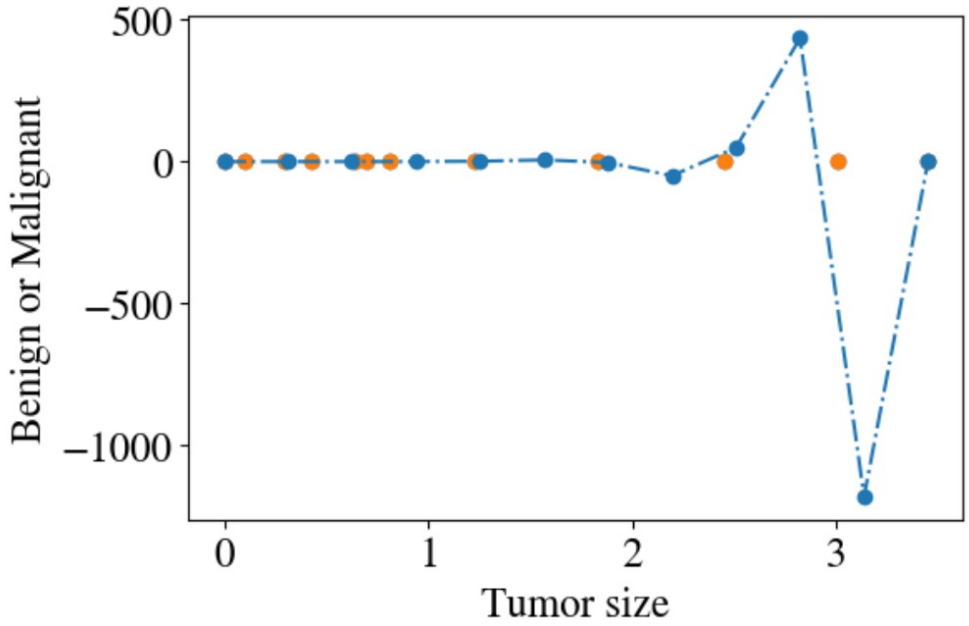


15^{th} order polynomial fit

- Looks great!



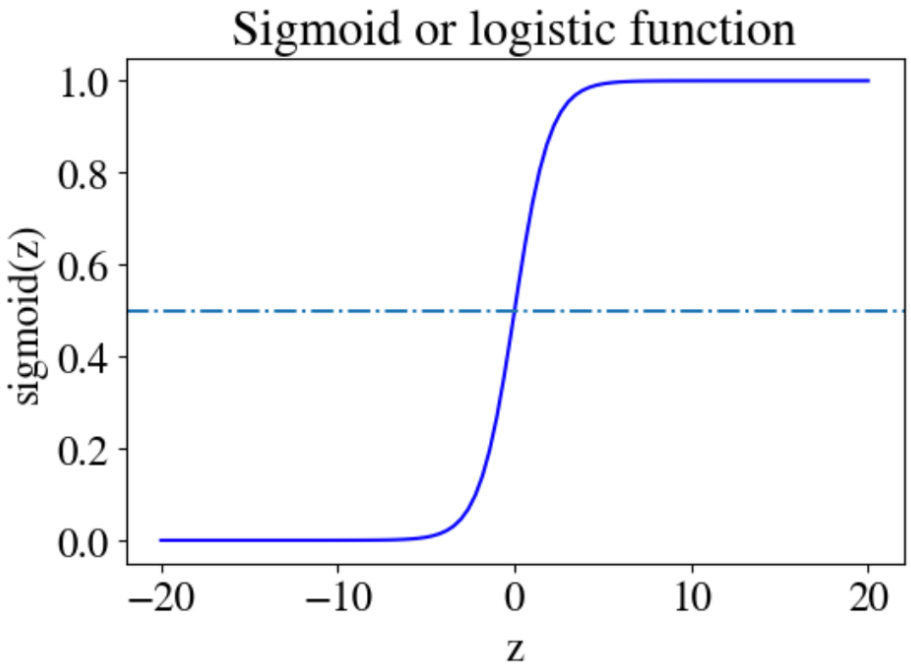
- Until we add few more measurements:



We need a new model hypothesis!

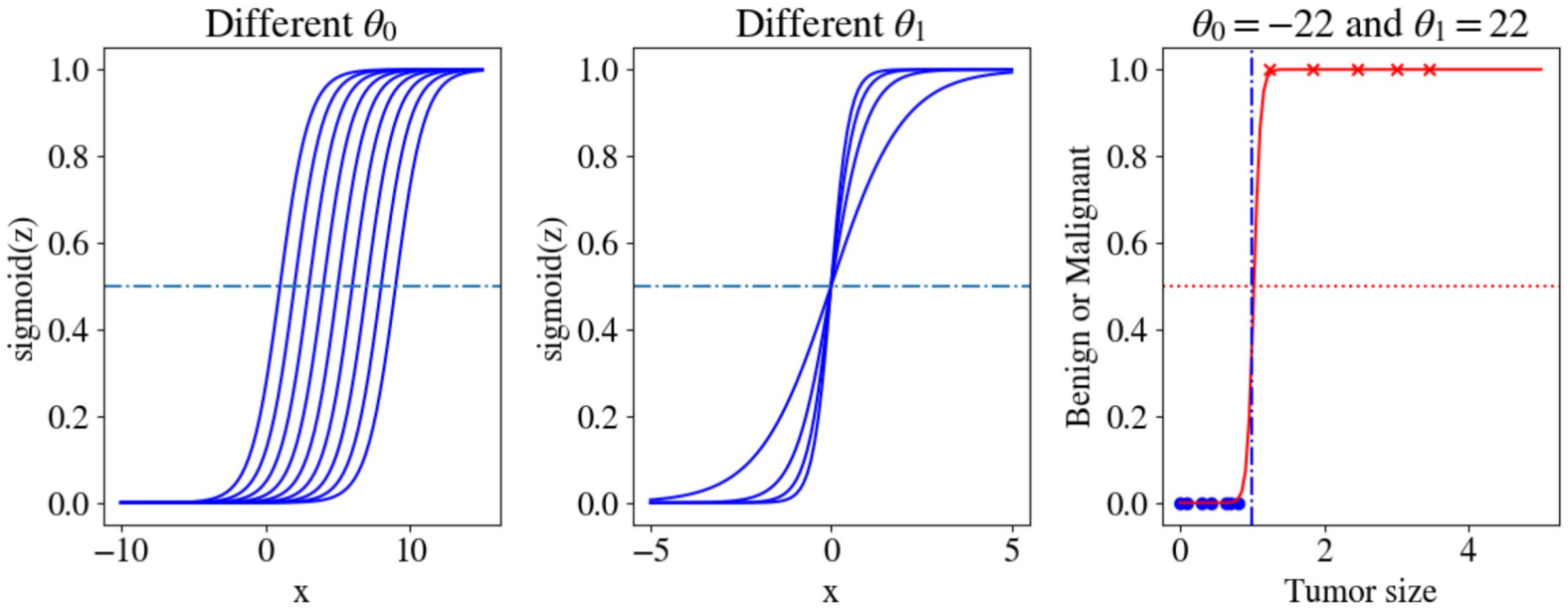
- Solution: **sigmoid** = **logistic function:** $y = g(z) = \frac{1}{1+e^{-z}}$
- $g(0) = 0.5$ for $z = 0$
- $g(z > 0)$ quickly approaches 1
- $g(z < 0)$ quickly approaches 0
- $0 < g(z) < 1$ for all $-\infty < z < \infty$
- Continuous probability around $z = 0$

- Fits the bill to predict binary *y*!



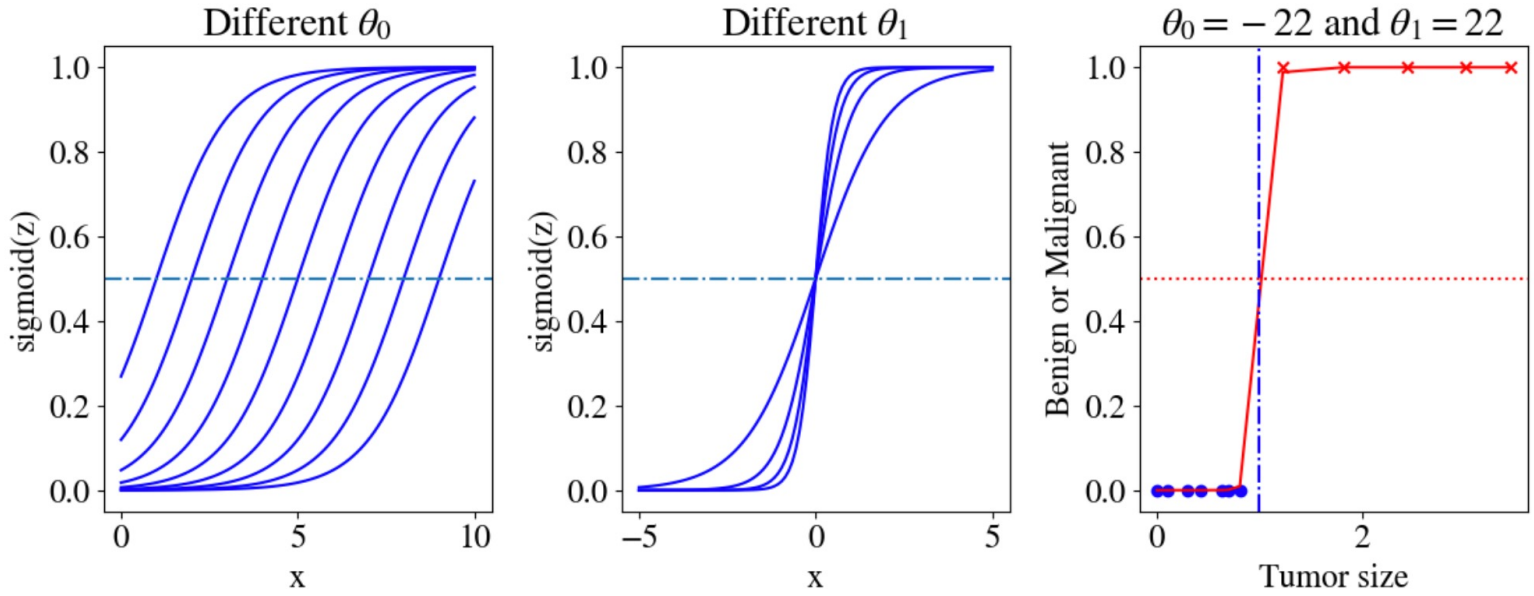
Logistic regression hypothesis

- We can tune (= fit) sigmoid by specifying $y = h(x;\theta) = g(\boldsymbol{X} \cdot \boldsymbol{\theta})$, e.g.:
- $y = \frac{1}{1+e^{-(\theta_0+\theta_1 x)}}$, θ_0 shifts the 'intercept', θ_1 changes slope in center



Logistic regression hypothesis

- Note that g(z) = 0.5 for $z = 0 = \theta_0 + \theta_1 x \Rightarrow x = -\frac{\theta_0}{\theta_1}$
- In this case, fit has $\theta_0 = -22, \theta_1 = 22$, so $g(z = 0) = g(x = 1) = 0.5$



Logistic regression hypothesis

- Note that g(z) = 0.5 for $z = 0 = \theta_0 + \theta_1 x \Rightarrow x = -\frac{\theta_0}{\theta_1}$
- In this case, fit has $\theta_0 = -22, \theta_1 = 22$, so $g(z = 0) = g(x = 1) = 0.5$
- Note, sigmoid *g*(*z*) is still continuous! Interpret as *probability* of *y* = 1.

- We can then threshold:

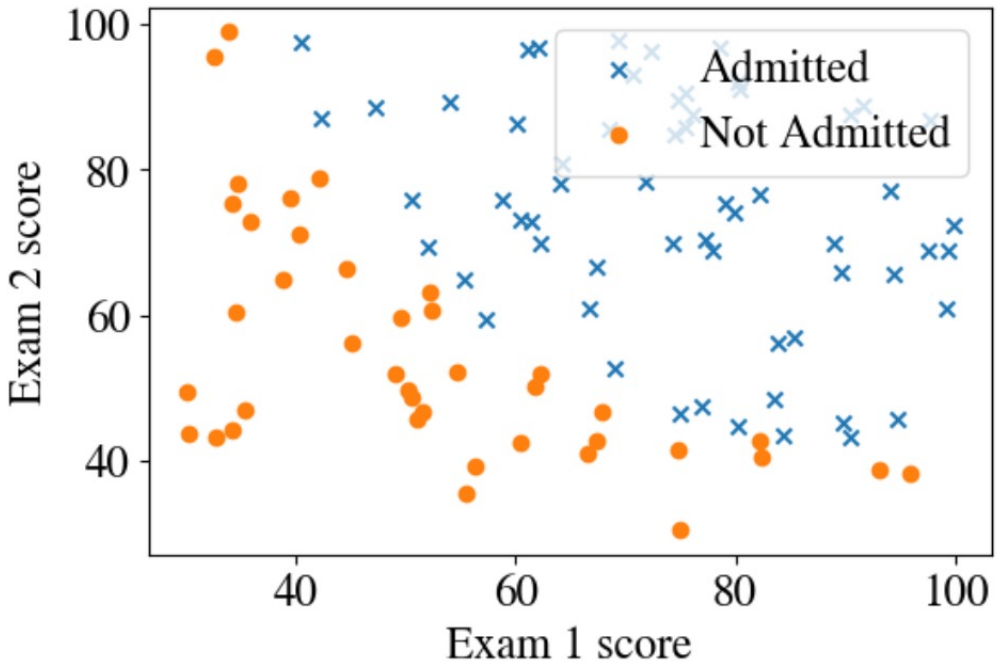
  y = 1 for $g(\boldsymbol{X} \cdot \boldsymbol{\theta}) \geq 0.5$ and *y* = 0 for $g(\boldsymbol{X} \cdot \boldsymbol{\theta}) < 0.5$
- In this case *y* = 1 for $x \geq 1$ (and vice versa)
- This relation defines the **decision boundary**.

Multivariate Linear Logistic Regression

Two features x_1 and x_2

- College admission (y = 1, stars) or rejection (y = 0, circles) based on 2 (continuous) exam scores

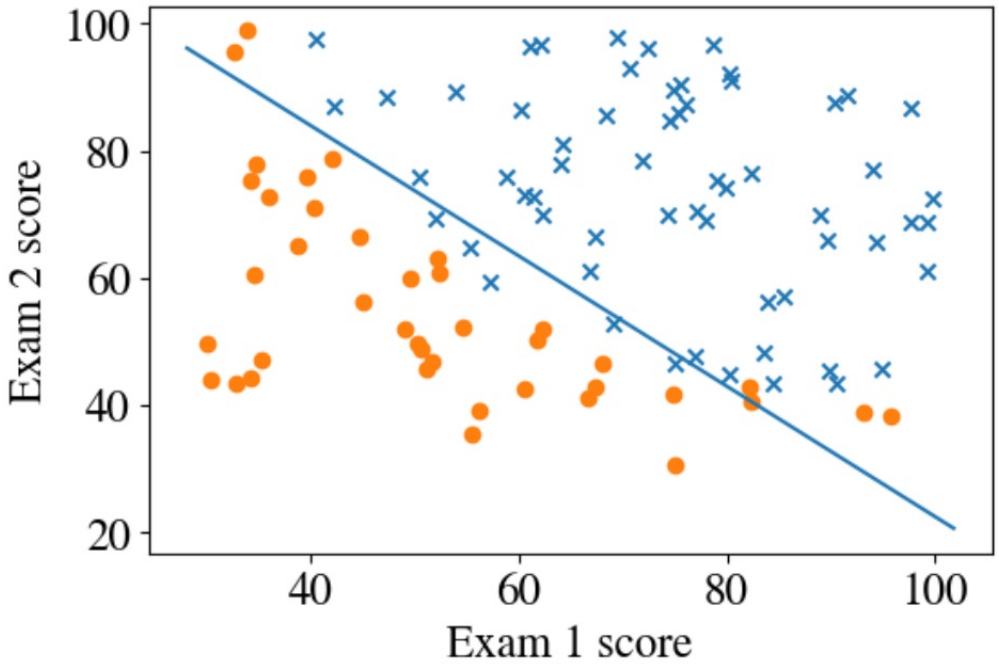
- Hypothesis:
- $h(x;\theta) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$



Fitting results

- Best for for $\theta_0 = -25, \theta_1 = 0.2, \theta_2 = 0.2,$
- Decision boundary for $\boldsymbol{X} \cdot \boldsymbol{\theta} = 0$:
- $0.2\ x_1 + 0.2\ x_2 = 25$ or
- $x_1 + x_2 = 125$

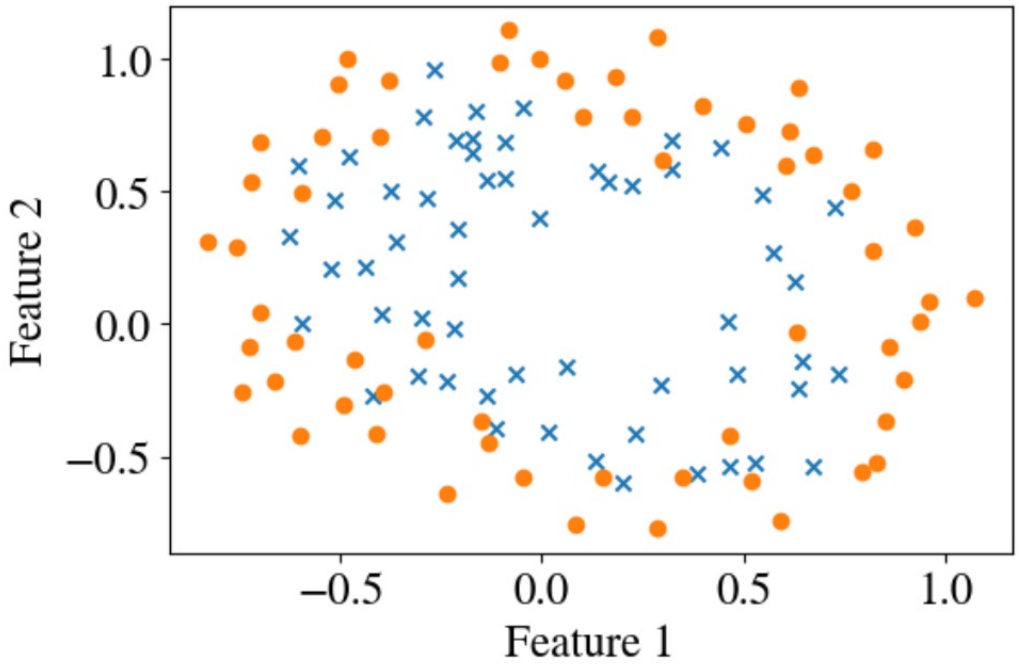
- This parameterizes shown line



Multivariate Non-Linear Logistic Regression

Non-linear data

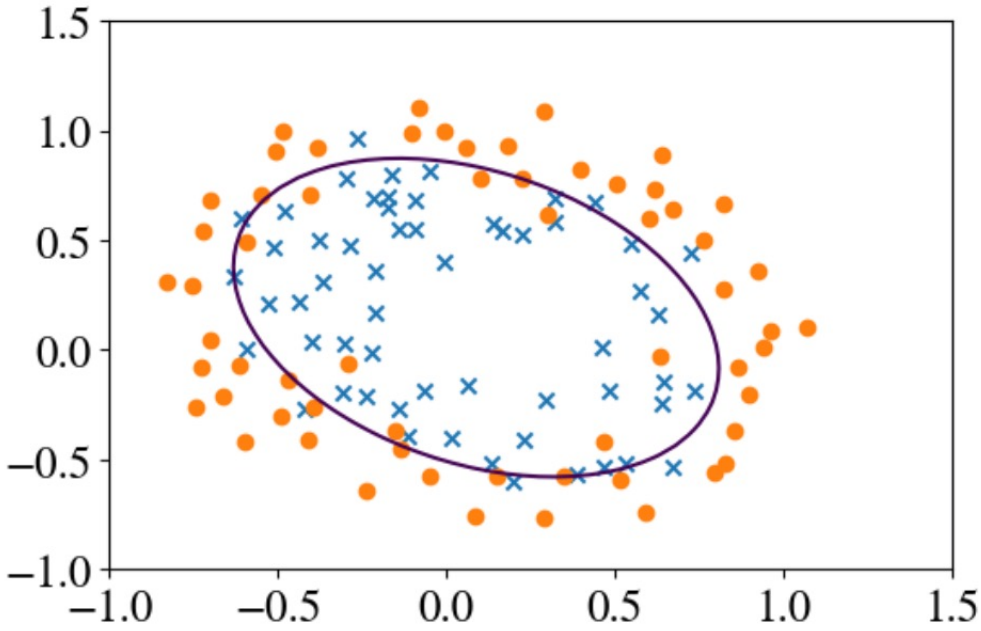
- Task: separate crosses for $y = 1$ from circles for $y = 0$
- Clearly, a straight line won't do
- Need non-linear model!



Same as for numerical regression!

- Try *n* polynomial features: $x_0 = 1$, x_1=(feature 1)^2 , x_2=(feature 2)^2, etc.
- Decision boundary given by:
- $\sum_{i=1}^{n} \theta_i x_i = 0$

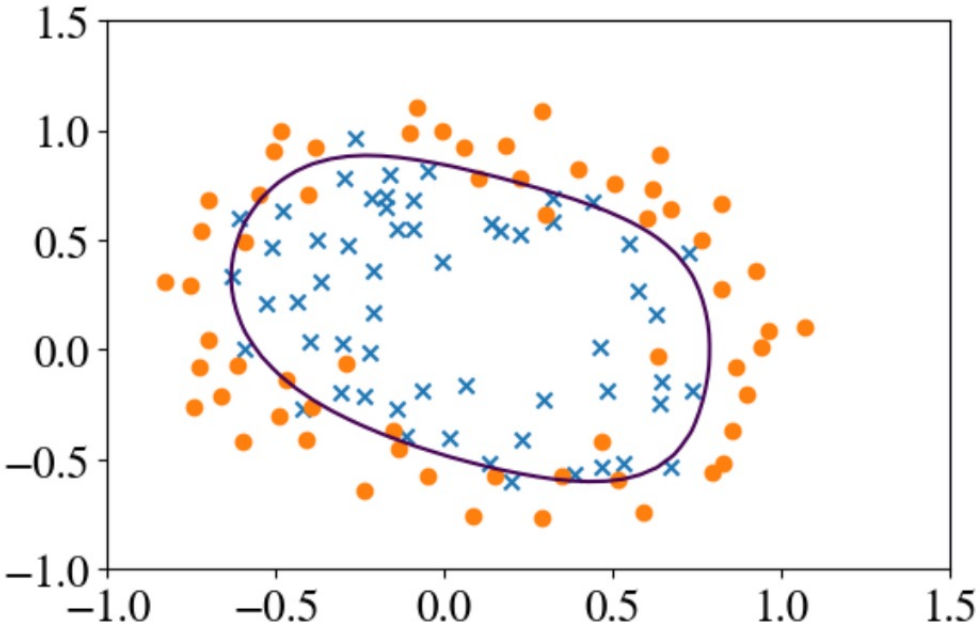
- For 2nd order, this gives an ellipse
- $\theta_1 x_1^2 + \theta_2 x_2^2 = -\theta_0$



Same as for numerical regression!

- Try *n* polynomial features: $x_0 = 1$, x_1=(feature 1)^2 , x_2=(feature 2)^2, etc.
- Decision boundary given by:
- $\sum_{i=1}^{n} \theta_i x_i = 0$

- 6th order polynomial

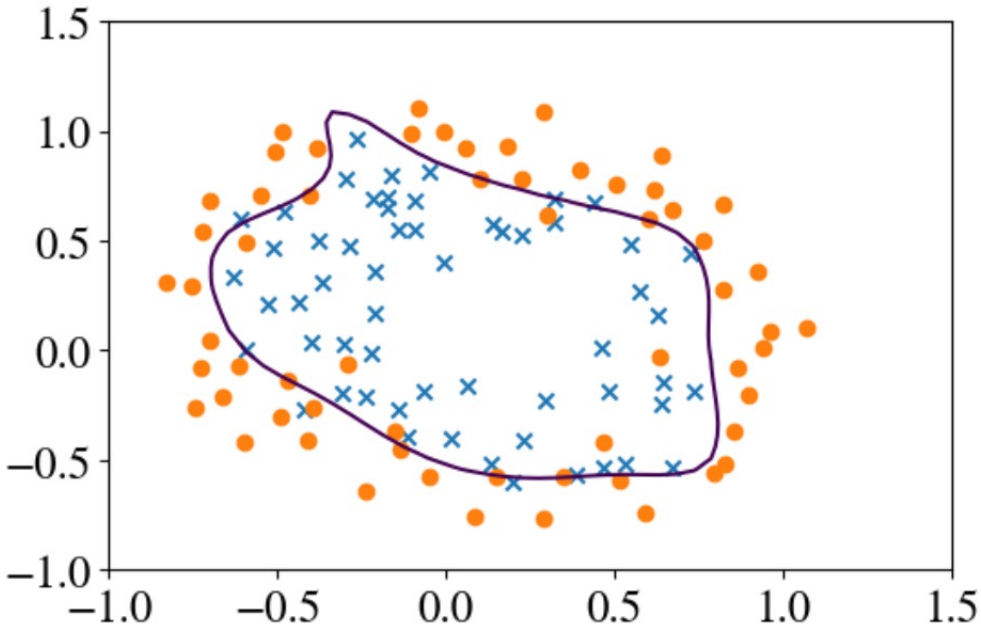


Same as for numerical regression!

- Try *n* polynomial features: $x_0 = 1$, x_1=(features 1)^2 , x_2=(features 2)^2, etc.
- Decision boundary given by:
- $\sum_{i=1}^{n} \theta_i x_i = 0$

- 12th order polynomial

- Overfitting!

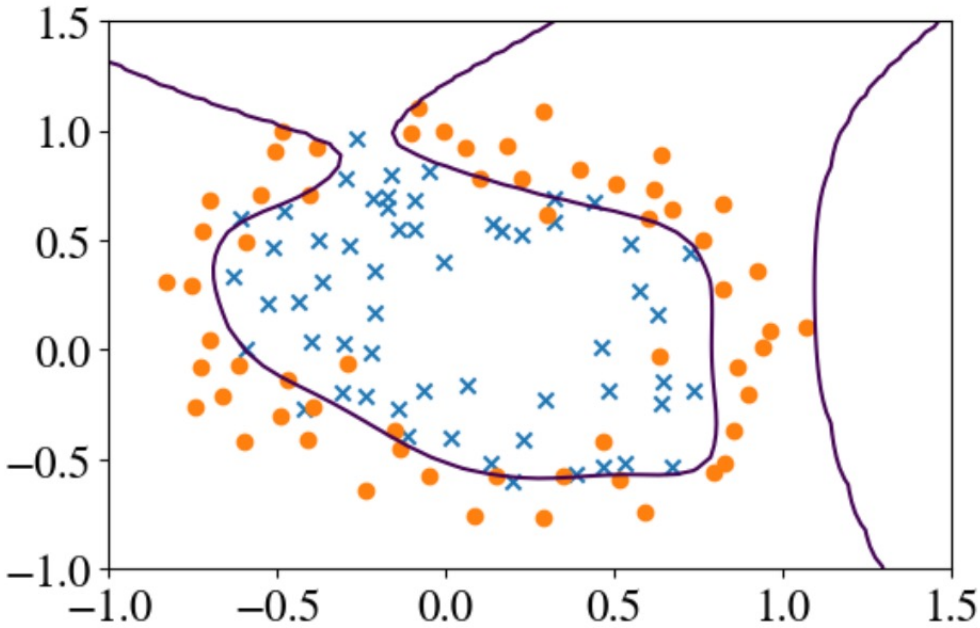


Same as for numerical regression!

- Try *n* polynomial features: $x_0 = 1$, x_1=(features 1)^2 , x_2=(features 2)^2, etc.
- Decision boundary given by:
- $\sum_{i=1}^{n} \theta_i x_i = 0$

- 24th order polynomial

- Overfitting!

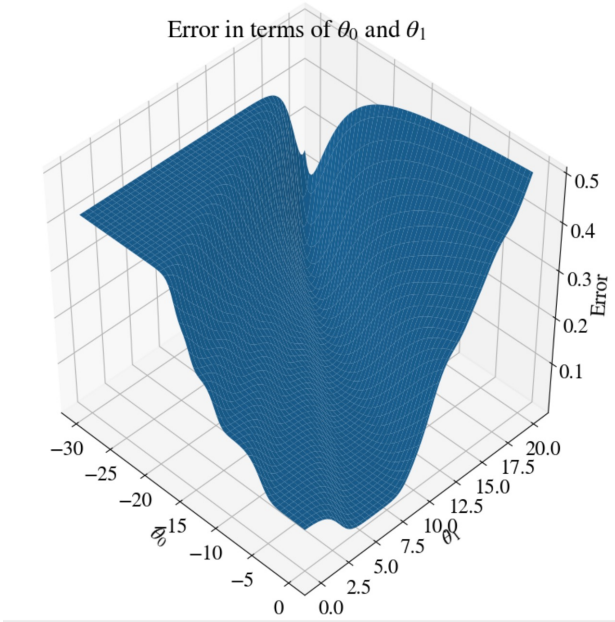


So how do we *find* best fit?

Error = Cost Function

- Mean-Squared-Error MSE would be:
- MSE $= \frac{1}{m}\sum_{i=1}^{m}(h(x;\theta) - y_{\text{true}})^2 = \frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{1+e^{-\sum \theta_j x_{j,i}}} - y_{\text{true}}\right)^2$

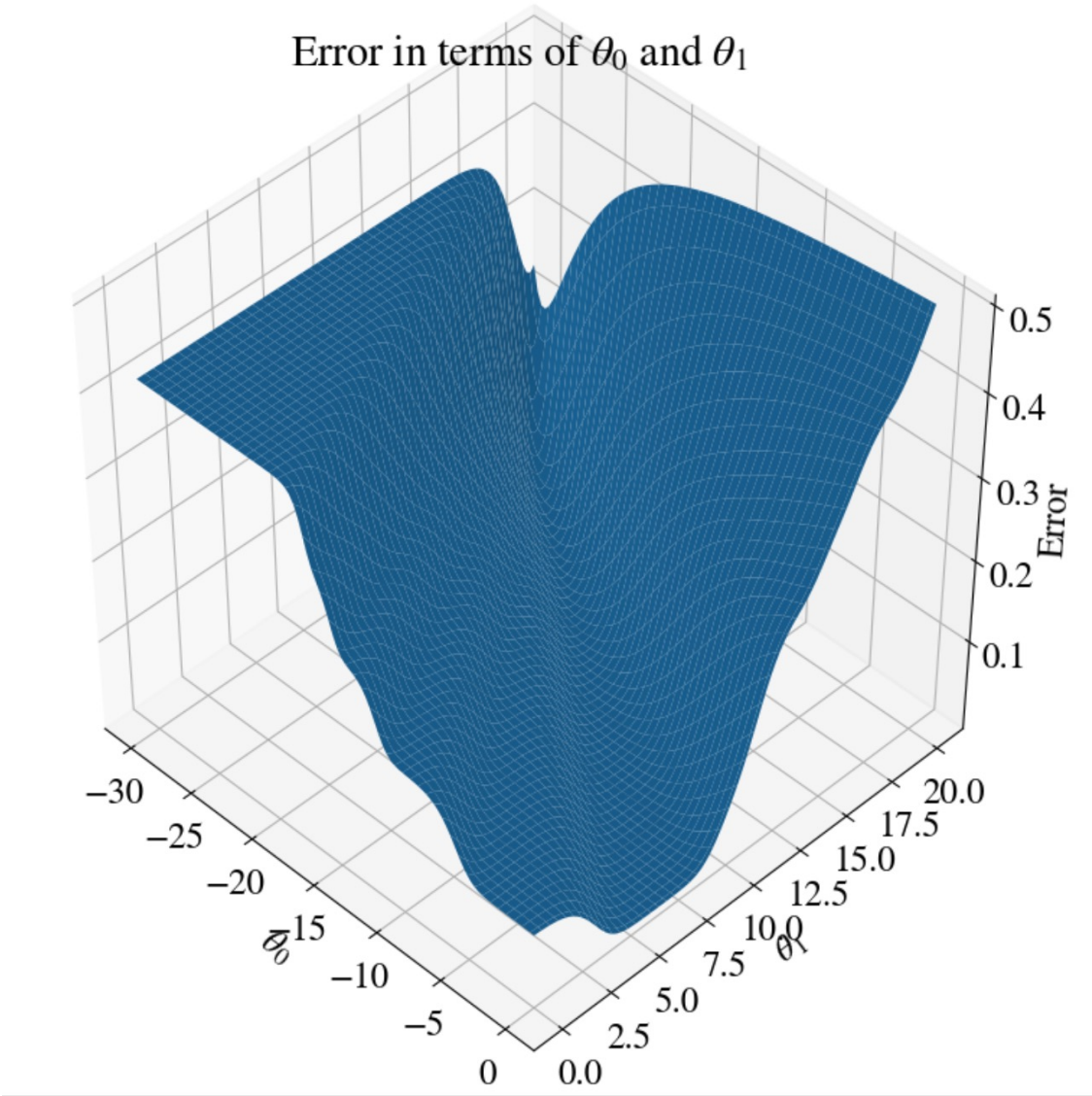
- Messy!



Not convex

- Many *local* minima,
- Will not converge to global minimum

- We need different measure of accuracy/error, i.e., a different cost function.



Cost function of logistic regression

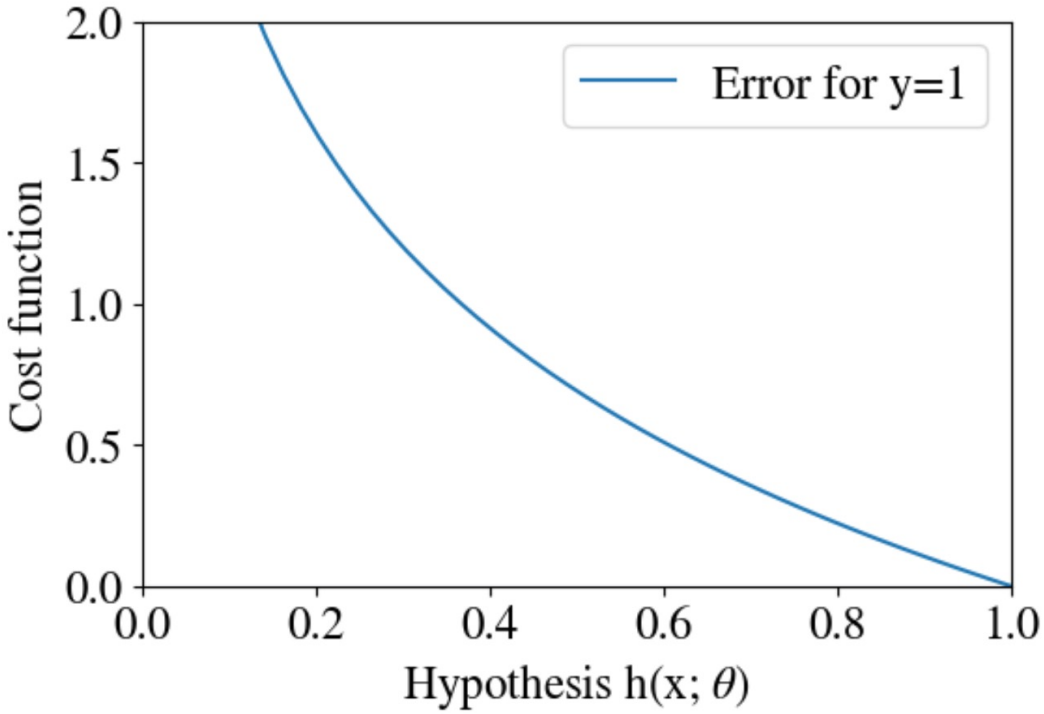
- Remember, true *y* can only be 0 or 1
- We define the error between a prediction/hypothesis given by $h(x;\theta) = g(\boldsymbol{X} \cdot \boldsymbol{\theta})$, and true values *y,* as:

$$\boldsymbol{J}(\theta) = \begin{cases} -\log(h(x,\theta)) & \text{if} \quad y = 1, \\ -\log(1 - h(x,\theta)) & \text{if} \quad y = 0. \end{cases}$$

What does this look like?

- If true label $y = 1$, and hypothesis $h(x;\theta) = g(\boldsymbol{X} \cdot \boldsymbol{\theta}) \approx 1$ then $J(\theta) \approx 0$ ✓
- $y = 1$ and $h(x;\theta) \ll 1$, then error $J(\theta) \uparrow +\infty$ ✓
- Very large error for wrong prediction

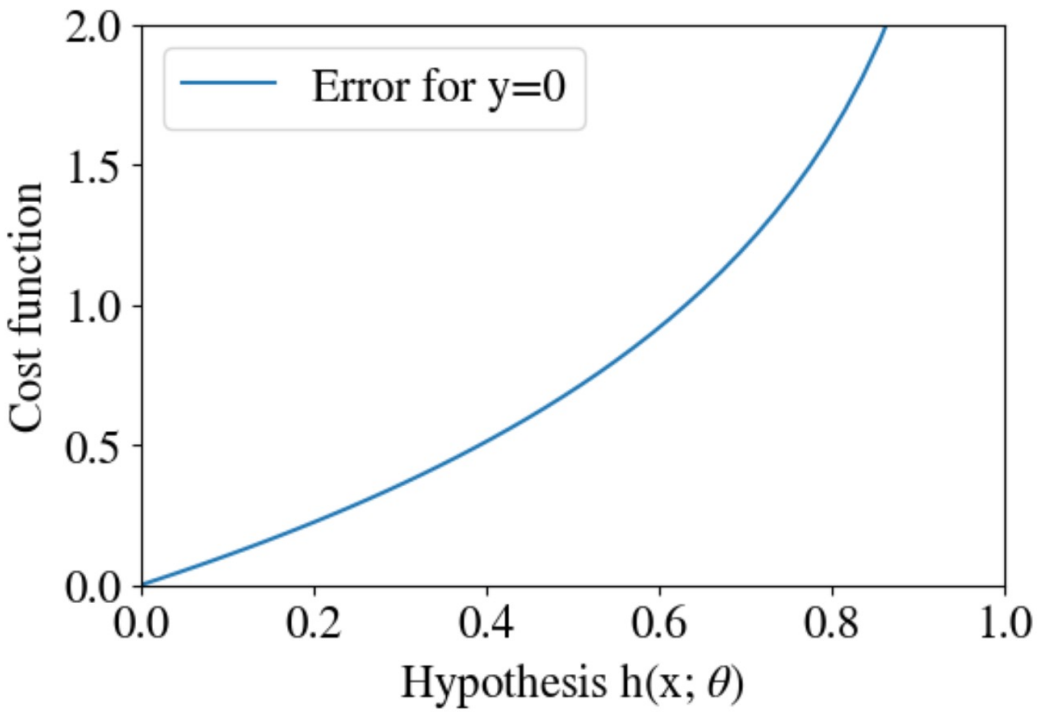
$$J(\theta) = \begin{cases} -\log(h(x,\theta)) & \text{if} \quad y = 1, \\ -\log(1 - h(x,\theta)) & \text{if} \quad y = 0. \end{cases}$$



What does this look like?

$$J(\theta) = \begin{cases} -\log(h(x,\theta)) & \text{if} \quad y = 1, \\ -\log(1 - h(x,\theta)) & \text{if} \quad y = 0. \end{cases}$$

- If true label y = **0**, and hypothesis $h(x;\theta) = g(\boldsymbol{X} \cdot \boldsymbol{\theta}) \approx 0$ then $J(\theta) \approx 0$ ✓
- y = 0 and $h(x;\theta) \gg 1$, then error $J(\theta) \uparrow +\infty$ ✓
- Very large error for wrong prediction



More elegant way to write cost function

- We can define (for a single point *x* and label *y*):

$$J(\theta) = -y\log\big(h(x;\theta)\big) \;-\; (1-y)\log(1-h(x;\theta))$$

- Because *y* can only be 0 or 1.
- For *y* = 1 the second term is zero
- For *y* = 0, the first term is zero

- For *m* points:

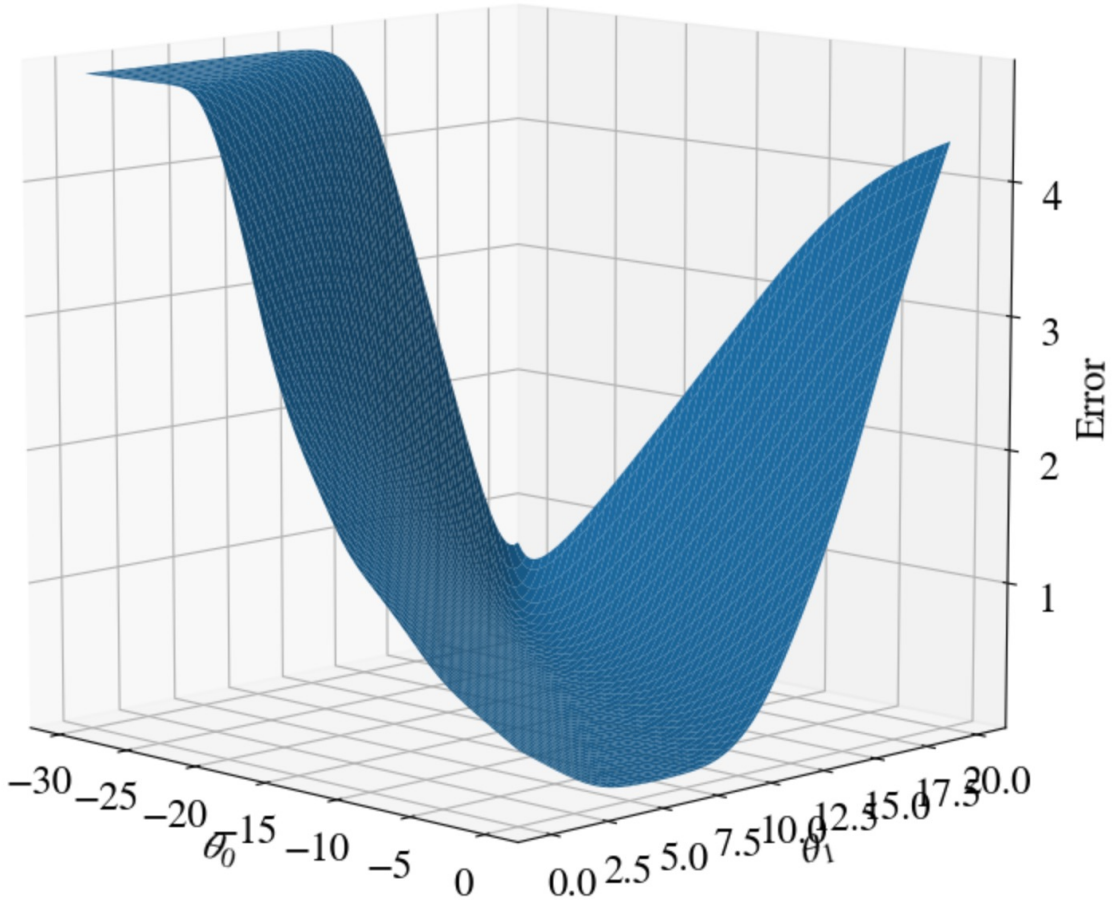
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^i\log\Big(h\big(x^i;\theta\big)\Big) \;-\; \big(1-y^i\big)\log(1-h\big(x^i;\theta\big))]$$

Illustration of logistic cost function

- Shown for two features
- θ_0 and θ_1
- Convex!
- One minimum

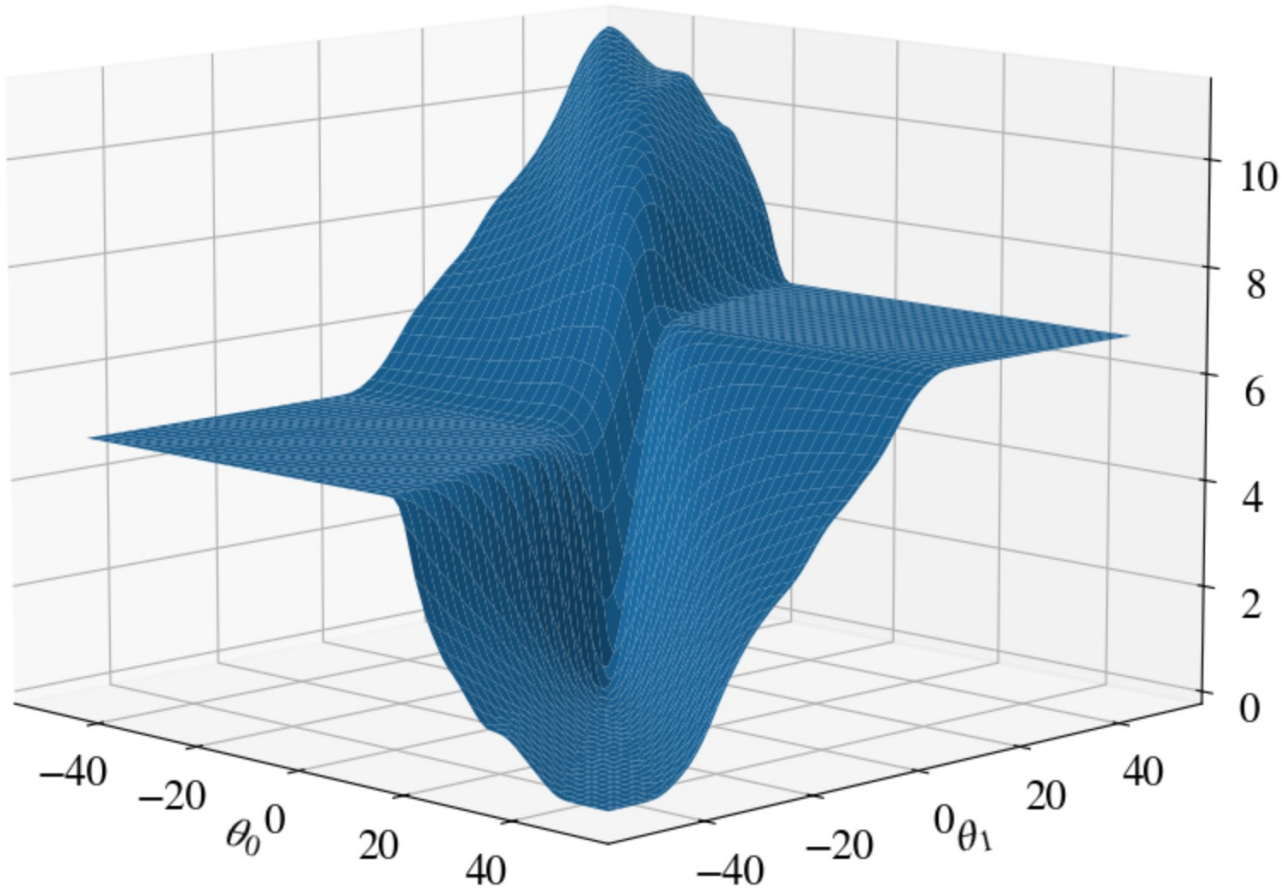
- Machine Learning task:

Find minimum of cost function



Subtlety

- “Vanishing gradients”
- When $|\theta_1|$ >> 0 and $|\theta_0| \gg 0$



Gradient Descent for Logistic Regression

Gradient Descent

- Just as for numerical regression, take small steps down the gradient of the error/cost function, using learning rate α

- $\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$ for each feature weight θ_j

- But what are the gradients/derivatives $\frac{\partial J(\theta)}{\partial \theta_j}$?

Cost Function Gradients

- $\frac{\partial J(\theta)}{\partial \theta} = -\frac{\partial}{\partial \theta}\frac{1}{m}\sum_{i=1}^{m}\left\{y^i \log \frac{1}{1+e^{-\theta \cdot x^i}} - \left(1-y^i\right)\log\left(1-\frac{1}{1+e^{-\theta \cdot x^i}}\right)\right\}$

- Looks like this would get messy…,

but surprisingly simple i.t.o. $h(x;\theta)$:

$$\frac{\partial J(\theta)}{\partial \theta} = -\frac{1}{m}\sum_{i=1}^{m}(\mathrm{h}(\mathrm{x}^i;\theta) - y^i)\, x^i$$

- Identical to numerical regression!!
- (but for different expression for the hypothesis, i.e., sigmoid)

Python Code for Logistic Regression

Hypothesis, Cost Function and its Gradient

```python
def sigmoid(z):
    return 1/(1 + np.exp(-z))

```

Add small ϵ to avoid taking log(0)

```python
def cost_logistic(theta, x, y):
    eps = 0.00001
    hypothesis = sigmoid(x.dot(theta))
    cost = - ( np.log(hypothesis+eps).dot(y) +  np.log(1-hypothesis+eps).dot(1-y)) / len(y)
    return cost
```

```python
def grad_logistic(theta,x,y):
    hypothesis = sigmoid(x.dot(theta))
    residual = (hypothesis-y)
    grad = residual.dot(x) / len(y)
    return grad
```

Vectorized Gradient Descent in while-loop
Identical to Numerical Regression!

```
nrsteps = 100000
nr_features = 2

thetas = np.zeros([nrsteps,nr_features+1])
errors = np.zeros(nrsteps)

m = len(y)
alpha = np.array([1,0.001, 0.001])

# Initial guess:
thetas[0,:] = initial_theta

i=0
converged = False
diverged = False
tolerance = 1e-4
while i < nrsteps-1 and not converged and not diverged:

    errors[i] = cost_logistic(thetas[i,:],x,y)
    thetas[i+1,:] = thetas[i,:] - alpha * grad_logistic(thetas[i,:],x,y)

    if i>0:
      improvement = np.abs(1- errors[i]/errors[i-1])
      if improvement < tolerance:
        converged = True
        print("Converged in", i, "iterations with error ", round(errors[i],2), "and thetas", np.round(thetas[i,:],
      elif improvement > 1:
        print(i, improvement, "Errors are diverging:", errors[i], ">", errors[i-1])
        diverged = True
    i+=1

```

Accelerated Gradient Descent

- Any minimization algorithms for numerical regression also work for logistic regression. Only need the cost function and gradient as input:

```
initial_theta = np.zeros(n + 1)
Nfeval=1
result = optimize.minimize(cost_logistic, initial_theta, args=(x,y), method='TNC', jac=grad_logistic, \
                           options={'disp': True}, tol=0.00001)


if result.success:
    besttheta = result.x
    nriterations = result.nit
    print("Converged after", result.nit, "iterations to best theta fits:", besttheta)
else:
    print("Accelerated method did not work, maybe try gradient descent.")
```

```
Converged after 11 iterations to best theta fits: [-24.70444573   0.2025209    0.19783393]
```

Accelerated Gradient Descent

- Any minimization algorithms for numerical regression also work for logistic regression. Only need the cost function and gradient as input:

```
[246]  1  fitted_theta = optimize.fmin_cg(cost_logistic, thetas,fprime=grad_logistic, args=(X,Y),gtol=0.001)
```

```
Optimization terminated successfully.
         Current function value: 0.001906
         Iterations: 7
         Function evaluations: 27
         Gradient evaluations: 27
```

Binary Prediction

- Our sigmoid hypothesis gives a continuous probability of $y = 1$
- To turn this into a binary classification, we can use simple threshold:

```
def predict(theta,x):
    hypothesis = sigmoid(x.dot(theta))
    p = hypothesis
    p[hypothesis >= 0.5]=1
    p[hypothesis < 0.5]=0
    return p
```

Non-Linear Logistic Regression

Non-Linear Logistic Regression

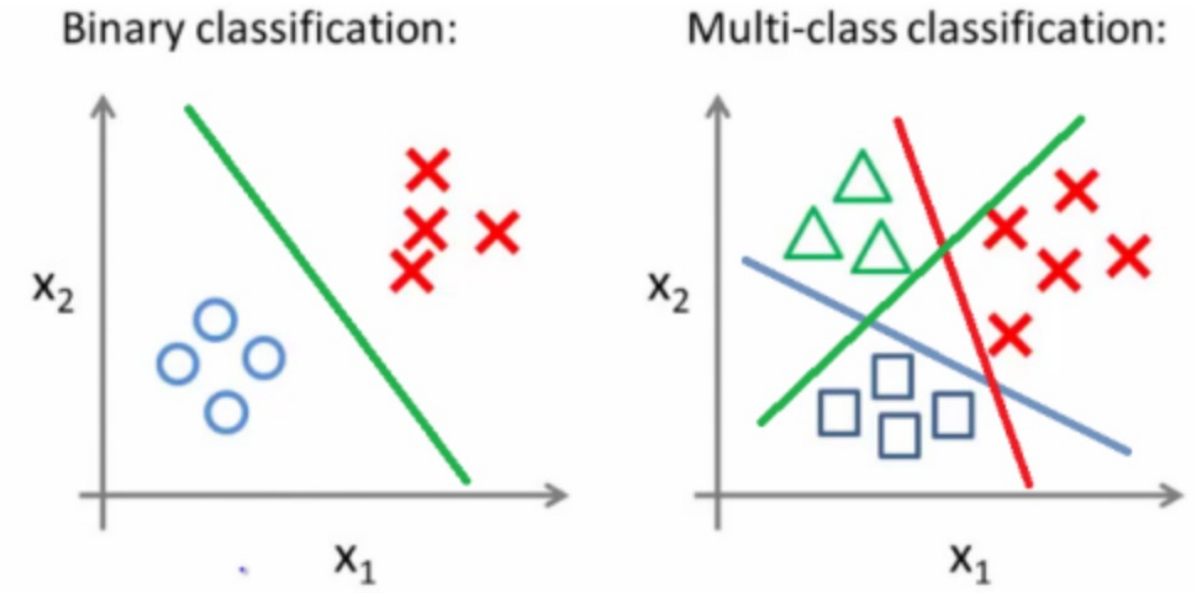
- Identical to generalization to non-linear *numerical* regression
- Simply define polynomial features and run linear logistic regression

- Gradient descent, accelerated GD, stochastic GD, mini-batch GD, all the same!

How about Multiple Classes??

One-vs-All Classification

- Surprisingly simple solution: loop over number of classes and solve/fit each as ‘yes’ ($y = 1$) or ‘no’ ($y = 0$) only for that class!



Conclusions

Conclusions

- Logistic regression **classifies** data into **discrete labels** *y*,
- Requires a different hypothesis: instead of fitting a polynomial function, we fit a **sigmoid** function *of* such a polynomial
- A different **logistic cost function** approaches 0 for correct prediction and infinity for wrong prediction
- Gradient of this cost function looks same as for numerical regression
- We can fit multiple classes with a ‘one-versus-all’ trick

- In terms of logistic cost function & gradient, all numerical regression machine learning methods work the same for classification!

Lecture Notes & Lab(s)

- Lecture notes:
  - ✓Week_4_Lecture_Logistic_Regression_Classification.pptx .pdf
  - ✓Week_4_Lecture_Logistic_Regression_Classification.ipynb

- Labs:
  - ✓Week_4_Lab_Logistic_Regression.ipynb