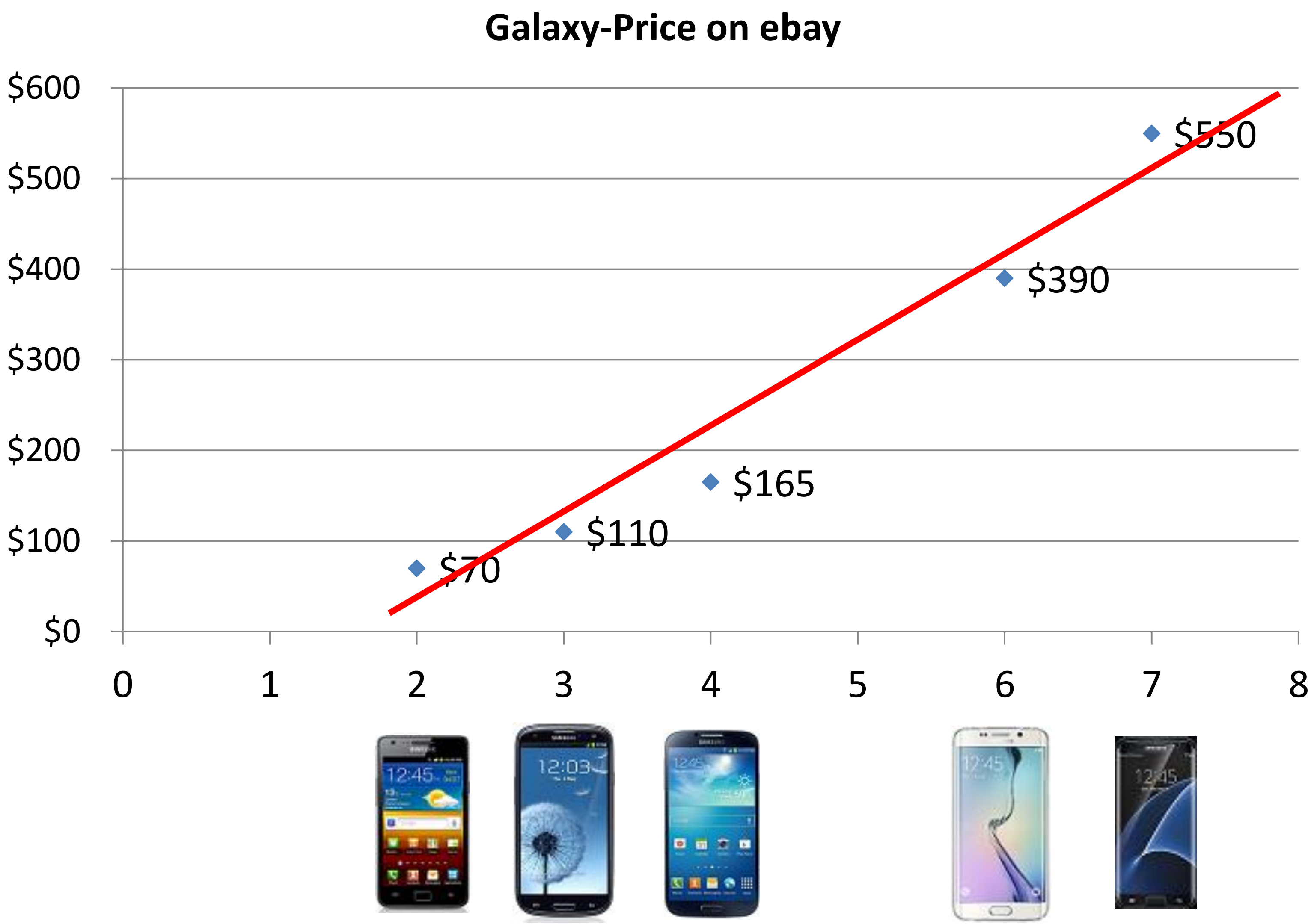
Linear and Logistic Regression

Amos Azaria

Estimating Galaxy-Phone Cost by Name



Formalizing Linear Regression

- y = wx + b
- What would we do if we had only 2 training examples?

We could solve 2 equations with 2 parameters

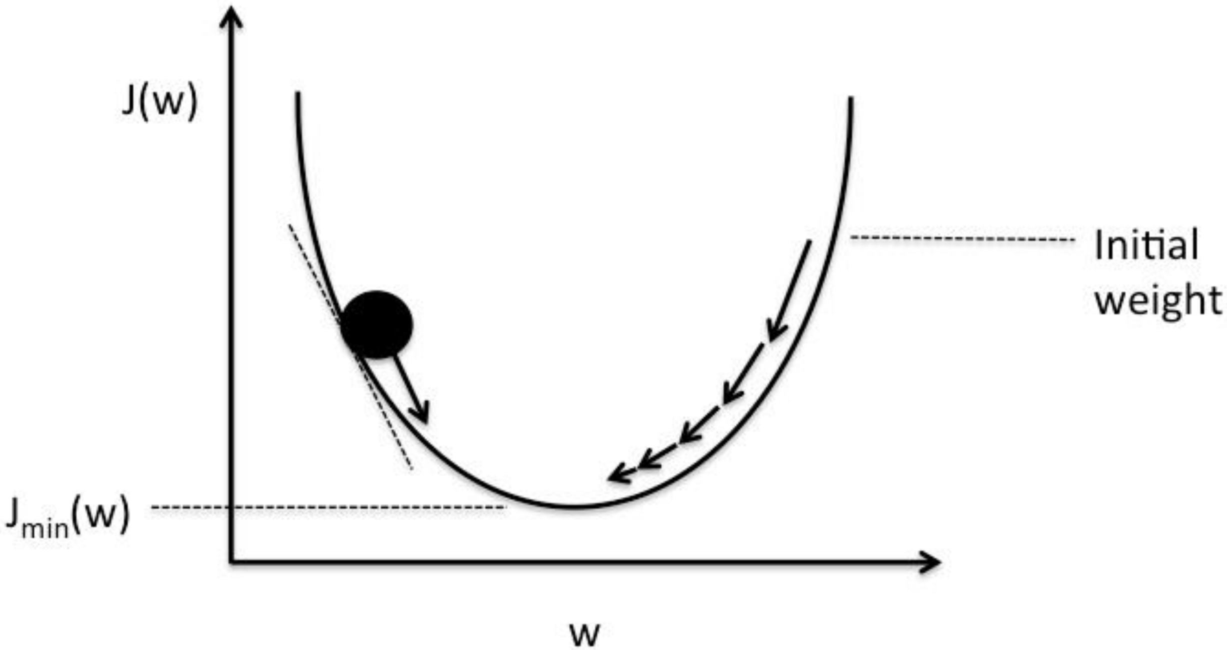
- Our prediction will be h(x) = wx + b
- $Y=\{y_1,y_2,y_3 \ldots y_m\}$, $X=\{x_1,x_2,x_3,\ldots x_m\}$
- Loss function: $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} |wx_i + b - y_i|$
- Or: $J(w,b) = \frac{1}{2m}\sum_{i=1}^{m} (wx_i + b - y_i)^2$

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m} (h(x_i) - y_i)^2$$

- Find w, b that will minimize J(w,b)

Gradient Descent

- Nc
  - w,b) has
  - erivative.



Schematic of gradient descent.

sets of f

nt update

What is the gradient (∇)?

- A vector which represents the derivation of a function, which has multiple parameters.
- Each entry is the function's derivative with respect to one of the parameters.

$$f_1(j_1, j_2) = 2j_1 \cdot j_2 + 7j_1$$

$$\nabla(f_1(j_1, j_2)) = (2j_2 + 7, 2j_1)$$

$$f_2(j_1, j_2, j_3) = 3j_1^2 j_2 j_3^3 + 5j_1 j_2$$

$$\nabla(f_2) = (6j_1 j_2 j_3^3 + 5j_2, 3j_1^2 j_3^3 + 5j_1, 9j_1^2 j_2 j_3^2)$$

The Gradient for Linear Regression

Our loss function: $J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(wx_i + b - y_i)^2$

$$\frac{\partial J}{\partial w} = \frac{1}{2m}\sum_{i=1}^{m} 2((wx_i + b - y_i)x_i)$$

$$= \frac{1}{m}\sum_{i=1}^{m}(wx_i + b - y_i)x_i$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(wx_i + b - y_i)$$

$$\nabla(J) = (\tfrac{1}{m}\sum_{i=1}^{m}(wx_i + b - y_i)x_i, \tfrac{1}{m}\sum_{i=1}^{m}(wx_i + b - y_i))$$

Gradient Descent in Linear Regression

- Pick random w, b
- Select the learning rate, α, (hyper-parameter), e.g. 0.01
- Repeat until convergence:
  - Update w to w-α$\frac{1}{m}\sum_{i=0}^{n} x_i(h(x_i) - y_i)$
  - Update b to b-α$\frac{1}{m}\sum_{i=0}^{n} 1 \cdot (h(x_i) - y_i)$

Linear Regression with SGD in Python
(Using the Galaxy Data-set)

```
import numpy as np
galaxy_data = np.array([[2,70],[3,110],[4,165],[6,390],[7,550]])
w = 0
b = 0
alpha = 0.01
for iteration in range(10000):
    deriv_b = np.mean(1*((w*galaxy_data[:,0]+b)-galaxy_data[:,1]))
    deriv_w = np.mean(galaxy_data[:,0]*((w*galaxy_data[:,0]+b)-galaxy_data[:,1]))
    b -= alpha*deriv_b
    w -= alpha*deriv_w
    if iteration % 200 == 0 :
        print("it:%d,  grad_w:%.3f, grad_b:%.3f, w:%.3f, b:%.3f" %(iteration, deriv_w, deriv_b, w, b))
print("Estimated price for Galaxy S5: ", w*5 + b)
```

Result:

it:0, grad_w:-1464.000, grad_b:-257.000, w:14.640, b:2.570

it:200, grad_w:-3.825, grad_b:19.693, w:70.598, b:-33.968

it:400, grad_w:-2.859, grad_b:14.720, w:77.230, b:-68.115

it:600, grad_w:-2.137, grad_b:11.003, w:82.188, b:-93.639

it:800, grad_w:-1.597, grad_b:8.224, w:85.893, b:-112.717

it:1000, grad_w:-1.194, grad_b:6.147, w:88.663, b:-126.977

it:1400, grad_w:-0.667, grad_b:3.434, w:92.280, b:-145.604

it:1800, grad_w:-0.373, grad_b:1.919, w:94.301, b:-156.010

it:2600, grad_w:-0.116, grad_b:0.599, w:96.062, b:-165.073

it:3600, grad_w:-0.027, grad_b:0.140, w:96.674, b:-168.226

it:4400, grad_w:-0.008, grad_b:0.044, w:96.802, b:-168.886

it:5400, grad_w:-0.002, grad_b:0.010, w:96.847, b:-169.116

it:6200, grad_w:-0.001, grad_b:0.003, w:96.856, b:-169.164

it:7000, grad_w:-0.000, grad_b:0.001, w:96.859, b:-169.179

it:7800, grad_w:-0.000, grad_b:0.000, w:96.860, b:-169.184

it:9800, grad_w:-0.000, grad_b:0.000, w:96.860, b:-169.186

Estimated price for Galaxy S5: 315.116281573

What would happen with alpha=0.5?

- Actual price was: $250
- Estimated price for Galaxy S1 is: -72.22 ☹

Adding Features

- Screen size
- Number of cores
- Core speed
- Memory size
- …

Adding Features (cont.)

- $x_1 = \{x_{11,}\ x_{12,}\ x_{13},\ldots,\ x_{1k}\}$
- $W = \{w_1,\ w_2,\ w_3,\ldots,w_k\}$
- Loss(W,b) = $\sum(\mathrm{y}-(\mathrm{Wx}+\mathrm{b}))^2$
- Sometimes we set: $x_{10,}\ x_{20,}\ x_{30,}\ldots,\ x_{n0} = 1$
  - => no need for “b”
- $W = \{w_0{=}b,\ w_1,\ w_2,\ \ldots\}$
- Loss = $\sum(\mathrm{y}-\mathrm{Wx}\ )^2$

Gradient Descent with Multiple Features (is actually the same…)

- ∇ is: $\sum_{i=0}^{n} \overleftarrow{x_i}\, (h(\overleftarrow{x_i}) - y_i)$
- Initialize W
- Repeat:
  - Calculate ∇: $\frac{1}{m} \sum_{i=0}^{n} \overleftarrow{x_i}\, (h(\overleftarrow{x_i}) - y_i)$
  - Update: W = W - α ∇

Adding quadratic feature

- We can add a feature which is simply a square of the first feature.
- We will end up with 2 weights and one bias:
  - $w_1 x + w_2 x^2 + b$
- This way, instead of fitting a linear line, we are actually fitting a quadratic function (parabola).

Quadratic Feature

```
import numpy as np
data_x = np.array([[2,4],[3,9],[4,16],[6,36],[7,49]])
data_y = np.array([70,110,165,390,550])
w1 = 0
w2 = 0
b = 0
alpha = 0.001
for iteration in range(1000000):
    deriv_b = np.mean(1*((w1*data_x[:,0]+w2*data_x[:,1]+b)-data_y))
    deriv_w1 = np.dot(((w1*data_x[:,0]+w2*data_x[:,1]+b)-data_y), data_x[:,0]) * 
1.0/len(data_y)
    deriv_w2 = np.dot(((w1*data_x[:,0]+w2*data_x[:,1]+b)-data_y), data_x[:,1]) * 
1.0/len(data_y)
    b -= alpha * deriv_b
    w1 -= alpha * deriv_w1
    w2 -= alpha * deriv_w2

print("Estimated price for Galaxy S5: ", np.dot(np.array([5,25]),np.array([w1, w2])) + b)
print("Estimated price for Galaxy S1: ", np.dot(np.array([1,1]),np.array([w1, w2])) + b)
```

Quadratic Feature (Weights as Vectors)

```
data_x = np.array([[2,4],[3,9],[4,16],[6,36],[7,49]])
data_y = np.array([70,110,165,390,550])

w = np.array([0.,0])
b = 0
alpha = 0.001
for iteration in range(1000000):
    deriv_b = np.mean(1*((np.dot(data_x,w)+b)-data_y))
    gradient_w = 1.0/len(data_y) * np.dot(((np.dot(data_x,w)+b)-data_y), data_x)
    b -= alpha*deriv_b
    w -= alpha*dradient_w

print("Estimated price for Galaxy S5: ", np.dot(np.array([5,25]),w) + b)
print("Estimated price for Galaxy S1: ", np.dot(np.array([1,1]),w) + b)
```

Results

Prediction for Galaxy S5: $263.63

(actual: $250) 😊

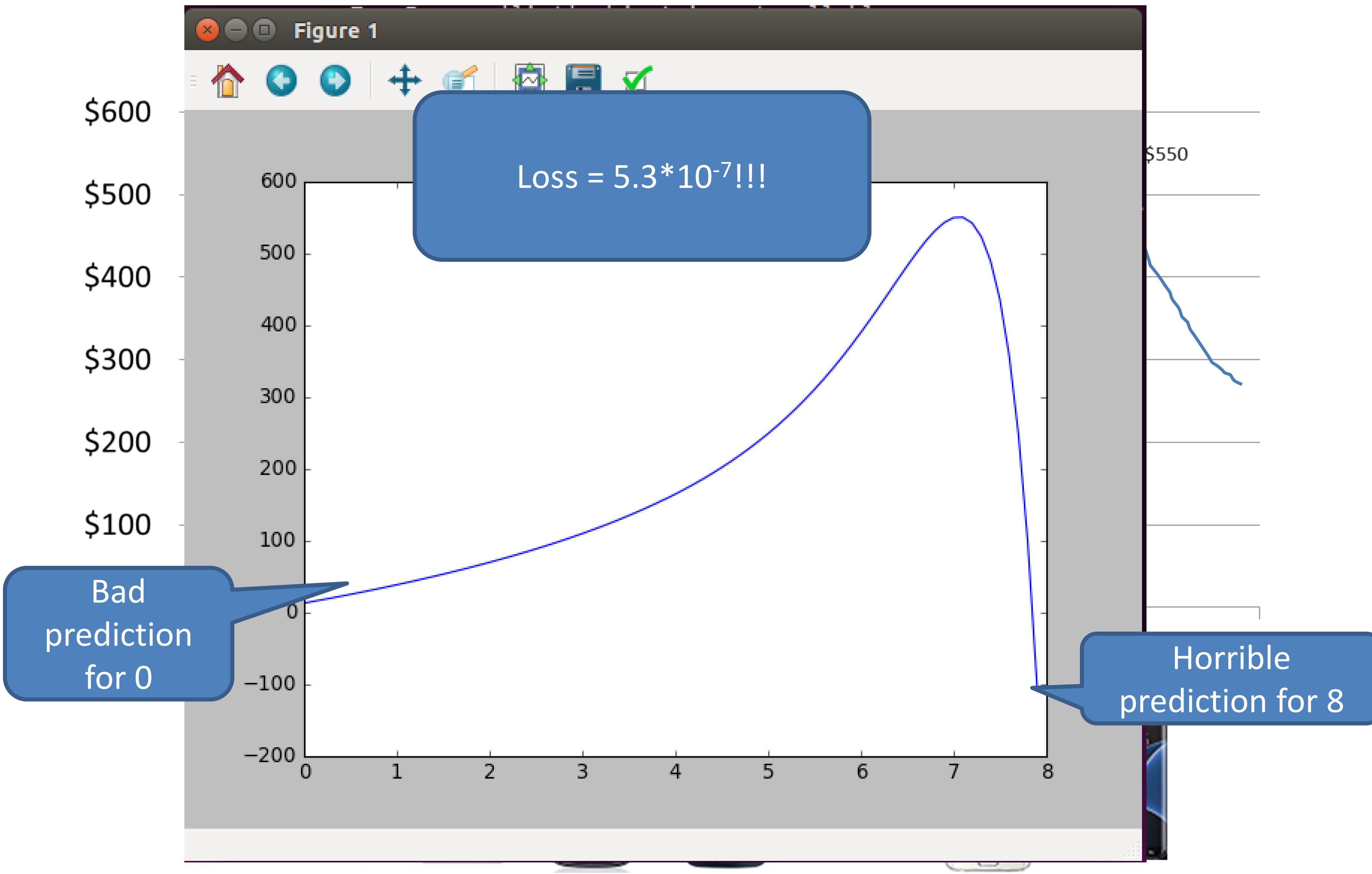
Prediction for Galaxy S1: $71.81

(actual: $30) 😔

Adding (too) Many Features

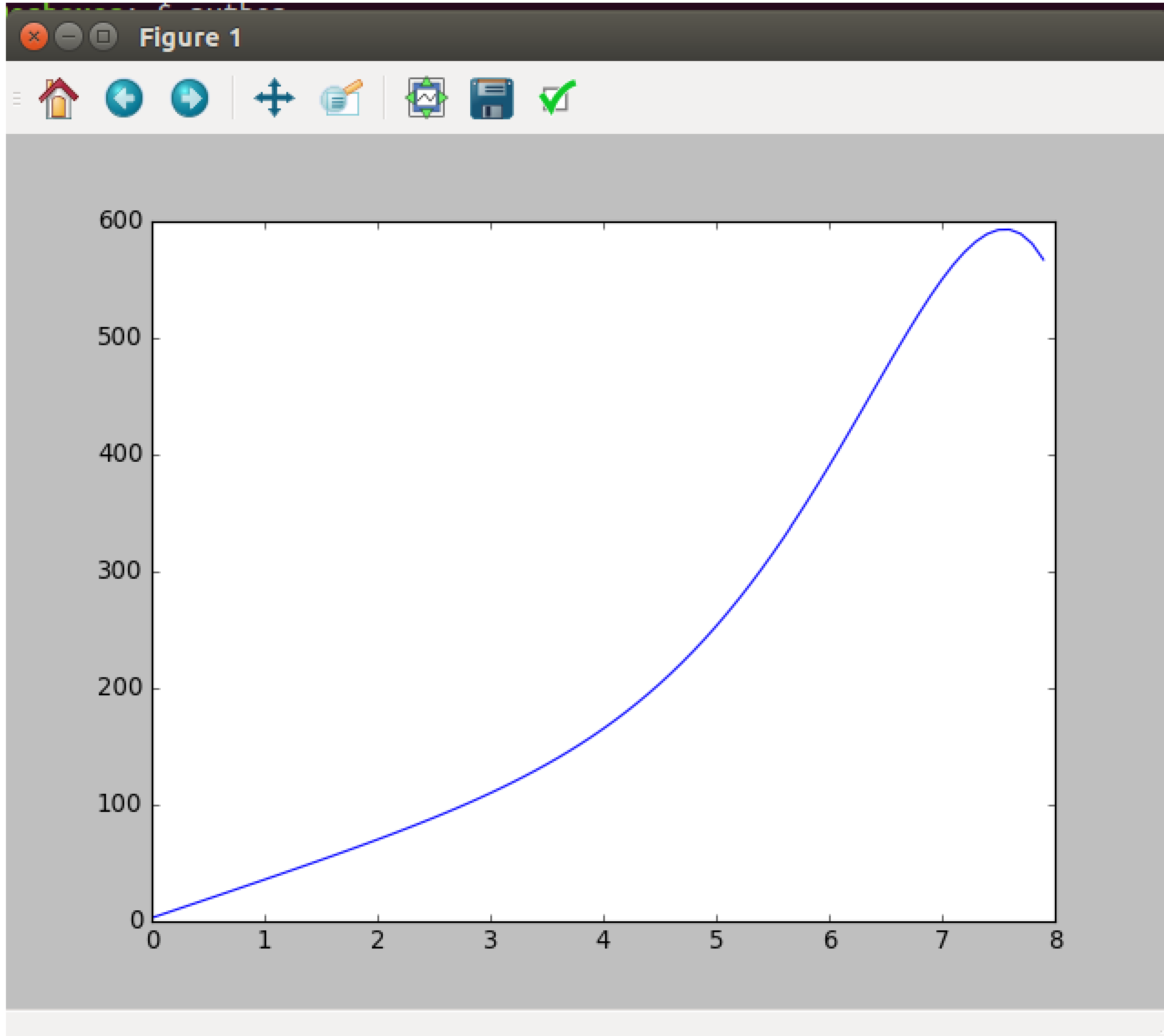
- We can add more and more features by adding additional polynomial terms (e.g. x^3, $x^{4,}$ etc.)
- (We need to make sure not to overflow, so we should really normalize the values we get).
- Should we add 20 features? What can happen?

Over-Fitting

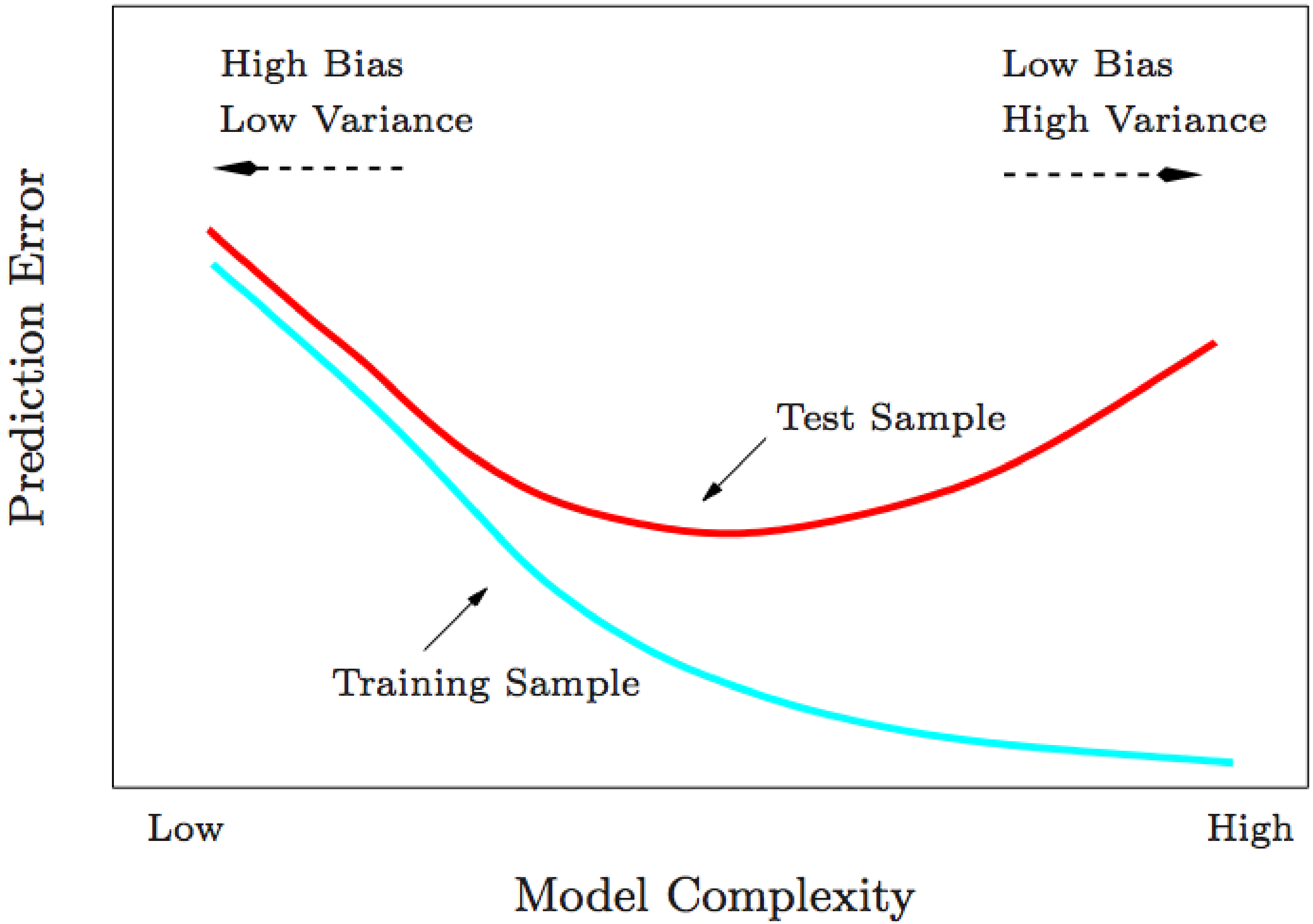


Results with "only" 10 features

Loss = 0.0049



Train/Test error



Credit: Collindcarroll.com

Train-Test-Validation

- If we want the test set to be a good prediction to what will happen with new data, it should be used only once.
- Therefore, we may want to have a validation set.
- The hyper-parameters / model complexity will be determined such that they maximize the accuracy of the validation set.

Closed form solution

- Linear regression has a closed-form solution:

$$W = (X^T X)^{-1} X^T Y$$

- We won't be focusing on it since we will be moving beyond linear regression to problems that do not have closed-form solution.
- Furthermore, the closed form solution may be less practical for big data.
- We will be using gradient descent (and its variants) until the end of the course.

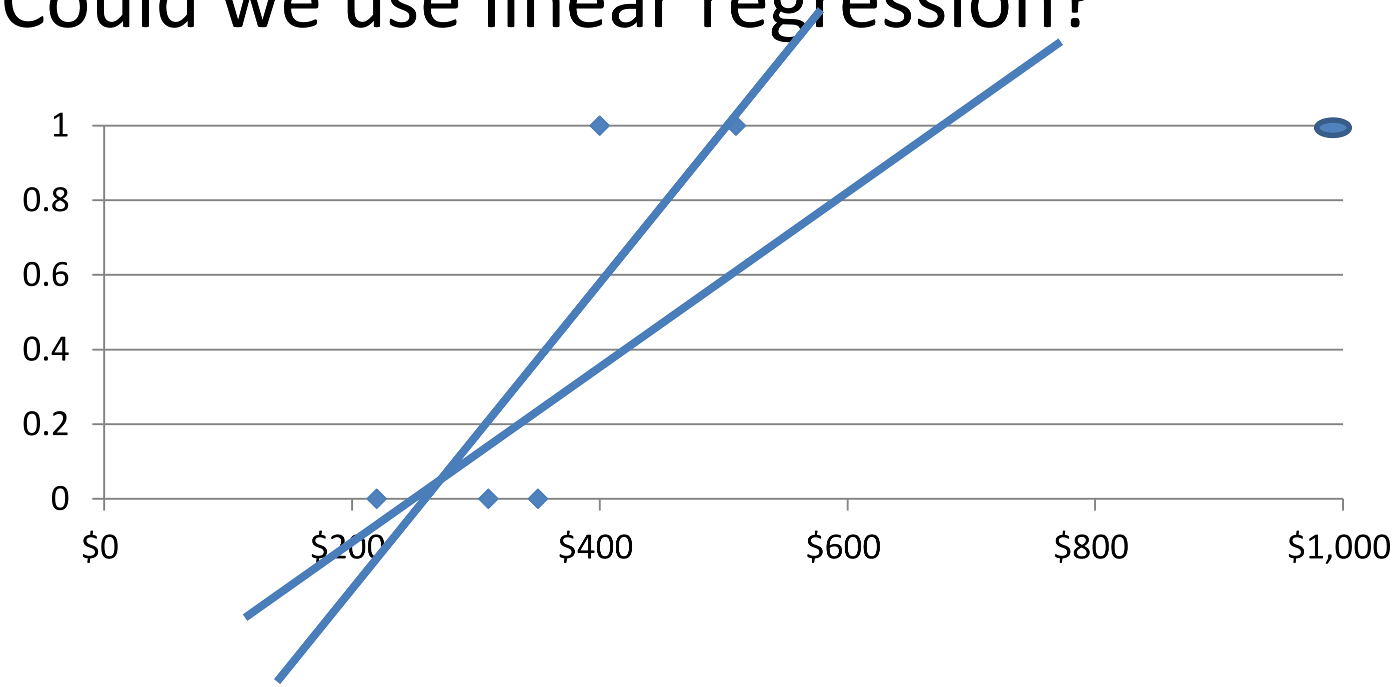
BGD, SGD, MB-GD

- Batch Gradient Descent (BGD): uses all Data-set to compute gradient (this is what we learnt).
  - May be too large, or take too long:
- Stochastic Gradient Descent (SGD): uses only a single example at a time (shuffle data first):
  - More iterations, since each iteration is less accurate
- Mini-Batch Gradient Descent (MB-GD): uses only a subset of the data-set, (e.g. 50) at a time:
  - A compromise which takes advantage of vectorization.

CLASSIFICATION

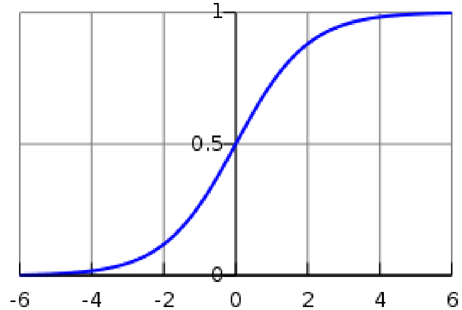
Classification

- Suppose we wanted to classify phones into new (1) or old/used (0), using the price as a single feature.
- Could we use linear regression?



Logistic Regression

- The Logistic function is:
  - $h(x) = \frac{1}{1+ e^{-(Wx+b)}}$



- A prediction of 1 will mean that we are **certain** that the value is 1.
- Instead of using least squares, we will use the following loss function:

This is actually the loss function we get when we try to maximize the likelihood of the data

$$J(w,b) = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i(log(h(x_i))) + (1-y_i)log(1-h(x_i))\right)$$

- It turns out that the Gradient of the loss is:
  - $\frac{1}{m}\sum_{i=0}^{n} x_i(h(x_i) - y_i)$

Gradient Descent in Logistic Regression

- Pick random w, b
- Select the learning rate, α, (hyper-parameter), e.g. 0.01
- Repeat until convergence:
  - Update w to w-α$\frac{1}{m}\sum_{i=0}^{n} x_i(h(x_i) - y_i)$
  - Update b to b-α$\frac{1}{m}\sum_{i=0}^{n} 1 \cdot (h(x_i) - y_i)$

How-come?

- Does the previous slide look familiar?
- The previous slide is identical to the slide we have seen in linear regression (I actually did a copy-paste and only changed the title).
- So how is logistic regression actually different than linear regression? If it is exactly the same algorithm, why not use linear regression?
- Obviously, the algorithms are different because the hypothesis (h(x)) is totally different…

Employed or not?

- We have a data-base with all our users and we want to send out job-offers.
- For that, we need to know all unemployed users, though we only know this information on a fraction of the data.
- We would like to build a classifier that determines whether a user is employed or not, based on the user's age, gender and years of experience.

Our dataset

This is fake data, sorry about any gender/age biases introduced intentionally…

- Employed users:
  - Female, 28 years old, 4 years of experience
  - Female, 60 years old, 34 years of experience
  - Female, 25 years old, 3 year of experience
  - Male, 54 years old, 20 years of experience
  - Male, 24 years old, 2 years of experience
  - Male, 39 years old, 12 years of experience
  - Male, 30 years old, 4 years of experience
- Unemployed users:
  - Female, 36 years old 10 years of experience
  - Female, 26 years old 1 year of experience
  - Male, 44 years old, 9 years of experience

Do you think that a female, 49 year old with 8 years of experience employed?

What about a male, 29 years old with 3 years of experience?

And if it were a female?

```
import numpy as np
data_x = np.array([[1,28,4],[1,60,34],[1,25,3],[0,54,20],[0,24,2],[0,39,12],[0,30,4],[1,3
data_y = np.array([1,1,1,1,1,1,1,0,0,0])

def h(x,w,b):
    return 1 / (1+np.exp(-(np.dot(x,w) + b)))

w = np.array([0.,0,0])
b = 0
alpha = 0.001
for iteration in range(100000):
    deriv_b = np.mean(1*( (h(data_x,w,b)- data_y)))
    deriv_w = np.dot((h(data_x,w,b) - data_y), data_x)*1/len(data_y)
    b -= alpha*deriv_b
    w -= alpha*deriv_w

print("User [1, 49, 8] prob of working: ", h(np.array([[1, 49, 8]]),w,b))
print("User [0, 29, 3] prob of working: ", h(np.array([[0, 29, 3]]),w,b))
print("User [1, 29, 3] prob of working: ", h(np.array([[1, 29, 3]]),w,b))
```

Results

User [1, 49, 8] prob of working: [0.21107079]

User [0, 29, 3] prob of working: [0.66430518]

User [1, 29, 3] prob of working: [0.43735087]

Meaning of Logistic Regression Result

- The result given by logistic regression is suppose to relate to the probability of the instance belonging to the class p(y=1 | X).
- Logistic regression is a discriminative model, that is, it tries to model p(y | X) directly.
- Recall that in the naïve Bayes classifier (which is a generative model), we used the Bayes rule, and therefore had to model:

  p(y) and p(X | y) (and p(X))

$$p(Y \mid X) = \frac{p(Y)p(X|Y)}{p(X)}$$

Multiple Classes

- Many times classification is into several labels. E.g. Classify an image to an object: cat/dog/airplane/sea/house
- We could build 5 different classifiers…
- More often we use one-hot vector representations for the labels. E.g.: cat = [1, 0, 0, 0, 0], sea = [0, 0, 0, 1, 0]
- **SoftMax**: as if we do logistic regression for each output alone and then scale:
  - $h(y = i \mid x) = \frac{e^{(W_i x + bi)}}{\sum_j^k e^{(W_j x + bj)}}$
  - Note that: $\sum_{i \in \{0,1,\ldots,k-1\}} h(y = i|x)$=1

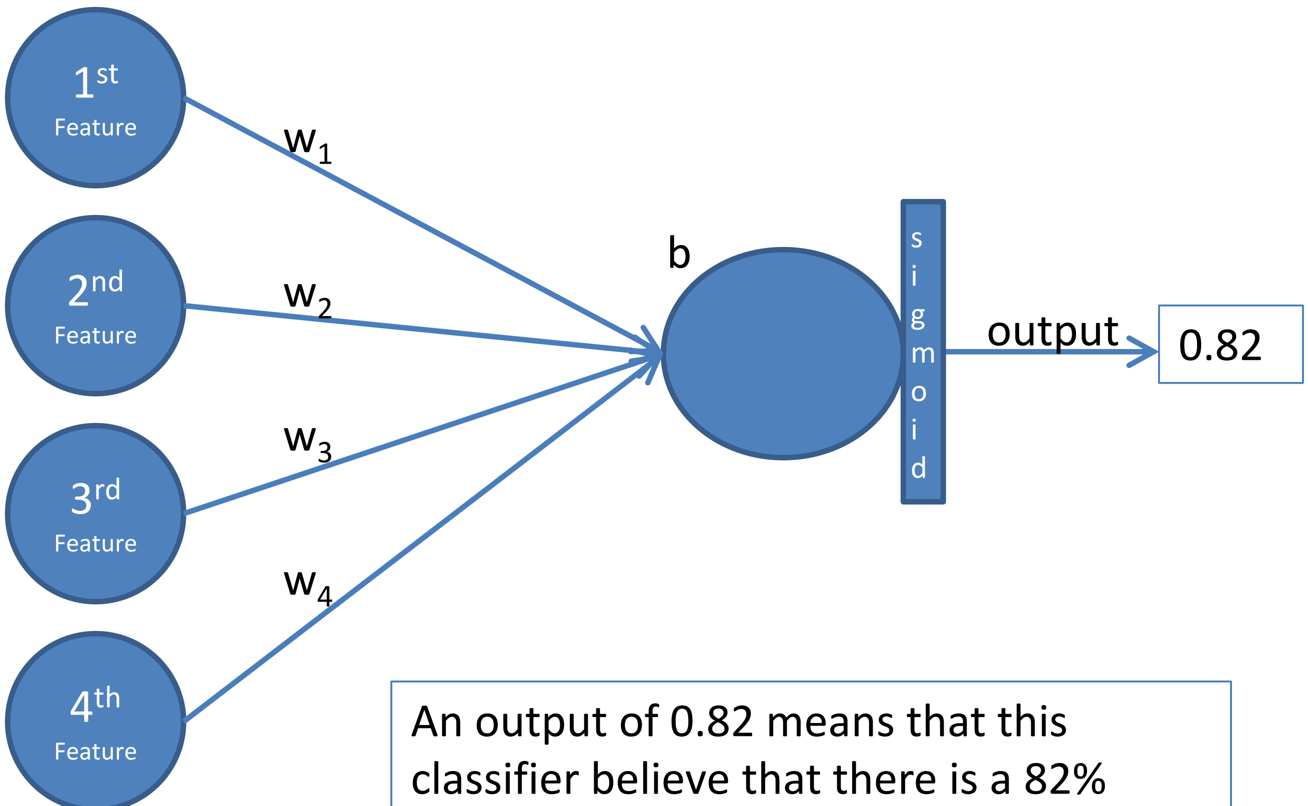
Average Cross Entropy Loss Function

- loss = -np.mean(y*np.log(h(x)))

Prediction (SoftMax)	Actual	Log(pred.)	Cross Entropy
[0.4, 0.5, 0.1]	[1, 0, 0]	[-0.92, -0.69, -2.3]	-0.92
[0.2, 0.7, 0.1]	[0, 1, 0]	[-1.61, -0.36, -2.3]	-0.36
[0.1, 0.2, 0.6]	[0, 0, 1]	[-2.3, -1.61, -0.51]	-0.51

Loss = 0.597

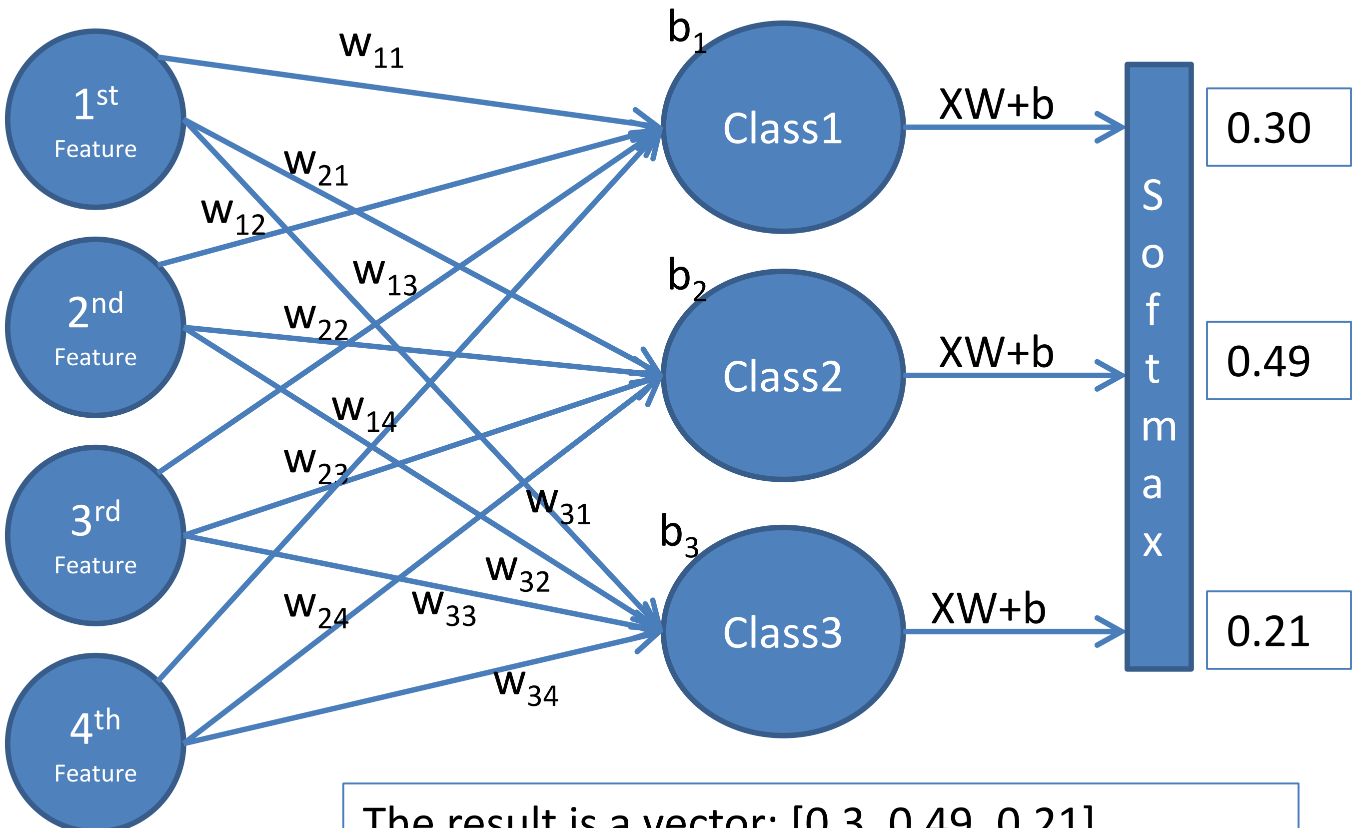
Visualization (Logistic Regression)



An output of 0.82 means that this classifier believe that there is a 82% chance that this instance is positive.

Therefore, this instance is classified as positive.

Visualization (SoftMax)



The result is a vector: [0.3, 0.49, 0.21], which should be interpreted as probabilities

Therefore, this instance was classified as Class2