EECS 738 – Machine Learning

Lab 3

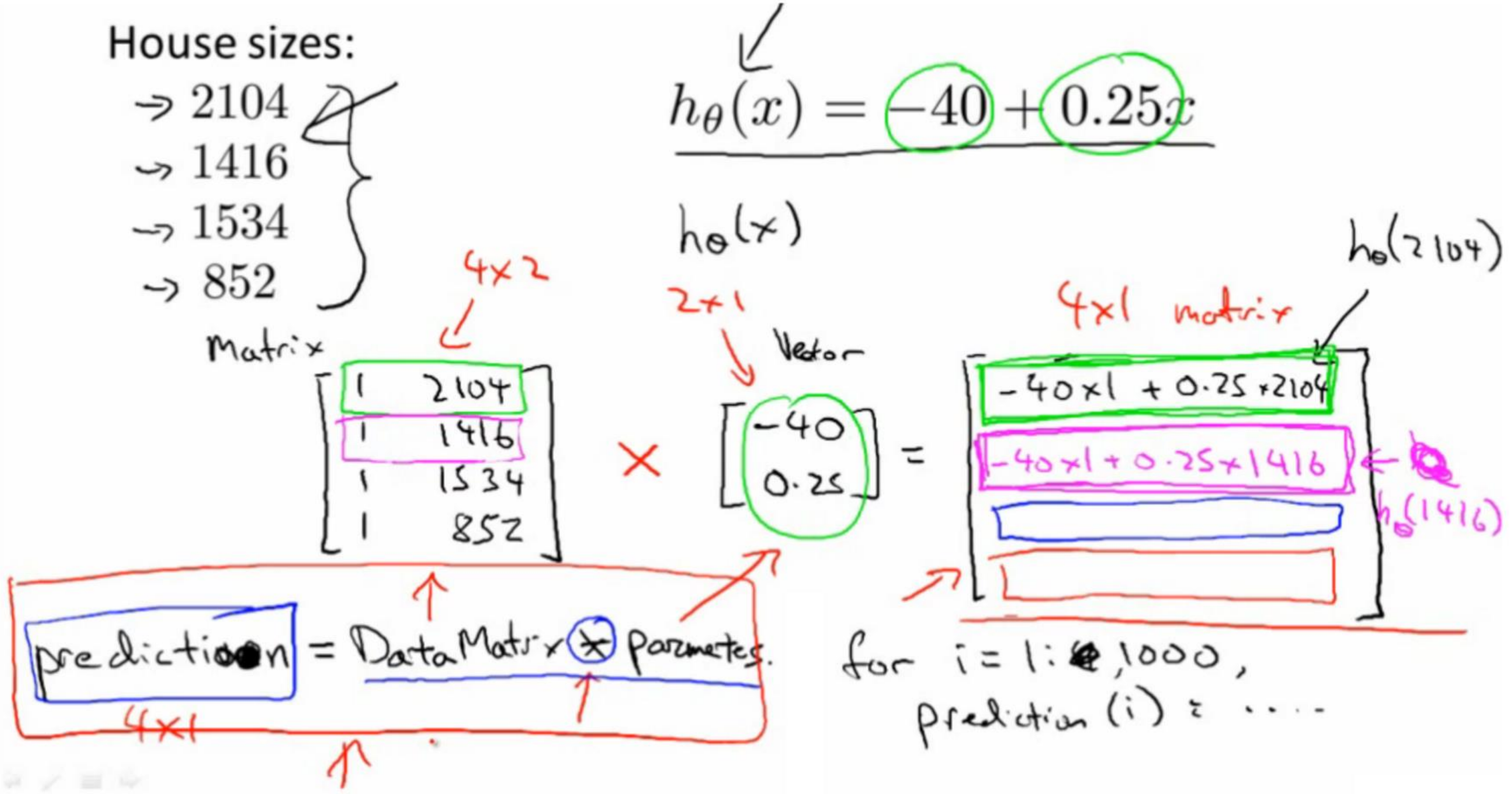
Linear and Logistic Regression

Announcements

- Everything related to the submission requirements is included in the comments in the code
- These slides are a review of the linear and logistic regression and the functions needed for completing the code

Simple linear Regression

- Prediction = Data Matrix * Parameters



Simple linear Regression

House sizes:

$$\begin{cases} 2104 \\ 1416 \\ 1534 \\ 852 \end{cases}$$

Have 3 competing hypotheses:

1. $h_\theta(x) = -40 + 0.25x$
2. $h_\theta(x) = 200 + 0.1x$
3. $h_\theta(x) = -150 + 0.4x$

Matrix

Matrix

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix} \times \begin{bmatrix} -40 & 200 & -150 \\ 0.25 & 0.1 & 0.4 \end{bmatrix} = \begin{bmatrix} 486 & 410 & 692 \\ 314 & 342 & 416 \\ 344 & 353 & 464 \\ 173 & 285 & 191 \end{bmatrix}$$

Prediction of first h_θ

Predictions of 2^{nd} h_θ

linear Regression with Multiple Features

- In original version we had:

X = house size, use this to predict

y = house price

Now we have more variables (such as number of bedrooms, number floors, age of the home)

Now we have multiple features

- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

For example

- $h_\theta(x) = 80 + 0.1x_1 + 0.01x_2 + 3x_3 - 2x_4$
    - An example of a hypothesis which is trying to predict the price of a house
    - Parameters are still determined through a cost function

Linear Regression
Cost Function (squared error)

- Fitting parameters for the hypothesis with gradient descent
    - Parameters are θ_0 to θ_n
    - Instead of thinking about this as n separate values, think about the parameters as a single vector (θ)
        - Where θ is n+1 dimensional
- Our cost function is

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Linear Regression
Gradient Descent

- We apply Gradient descent to find the minimum of the squared error cost function.
- “**Gradient descent** is a first-order iterative optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient (or of the approximate gradient) of the function at the current point.” (Wikipedia)
- This must have been be covered in class…

Gradient descent to Linear Regression

Gradient descent algorithm

$$\boxed{\begin{array}{l} \text{repeat until convergence } \{ \\ \quad \theta_j := \theta_j - \alpha \dfrac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \\ \qquad (\text{for } j = 1 \text{ and } j = 0) \\ \} \end{array}}$$

Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

Linear Regression
Gradient Descent

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update θ_j for $j = 0, \ldots, n$) }

- What's going on here?
  - We're doing this for each j (0 until n) as a simultaneous update (like when n = 1)
  - So, we re-set θ_j to
    - θ_j minus the learning rate (α) times the partial derivative of of the θ vector with respect to θ_j
    - In non-calculus words, this means that we do
      - Learning rate
      - Times 1/m (makes the maths easier)
      - Times the sum of
        - The hypothesis taking in the variable vector, minus the actual value, times the j-th value in that variable vector for EACH example

Linear Regression
#Iterations

- Number of iterations:

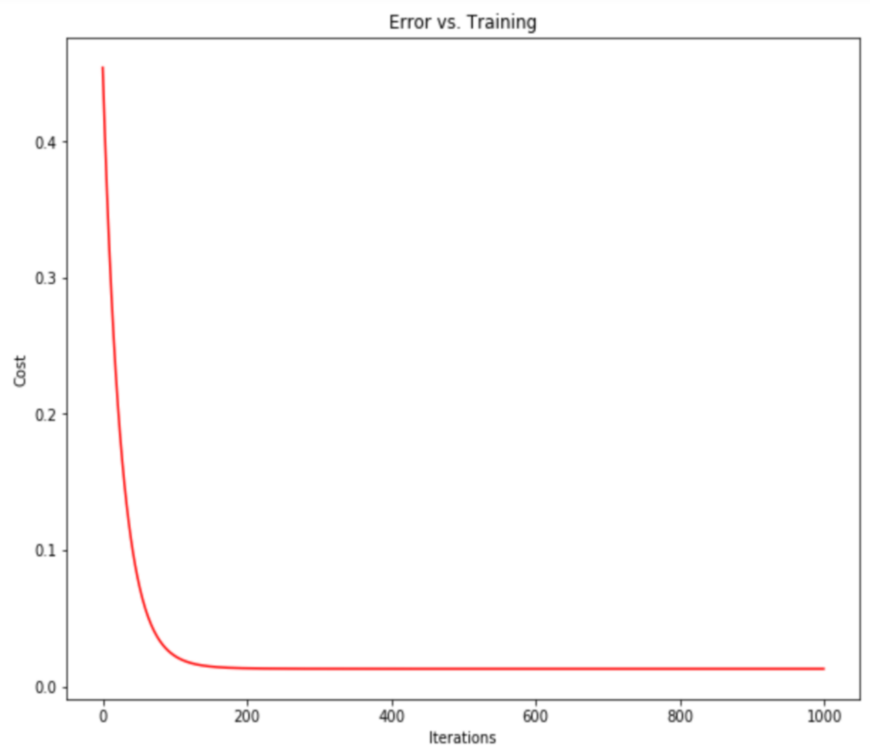
Number of iterations varies a lot 30 iterations

3000 iterations

3000 000 iterations

Very hard to tell in advance how many iterations will be needed

Can often make a guess based a plot like this after the first 100 or so iterations



Linear Regression

Learning Rate

alpha:
Rule a thumb regarding acceptable ranges
-3 to +3 is generally fine - any bigger bad
-1/3 to +1/3 is ok - any smaller bad

If you plot $J(\theta)$ vs iterations and see the value is increasing - means you probably need a smaller α

Generally:

- Try a range of alpha values
- Plot $J(\theta)$ vs number of iterations for each version of alpha
- Go for roughly threefold increases
  - 0.001, 0.003, 0.01, 0.03. 0.1, 0.3

Linear Regression
Normalization

- Can do **mean normalization**
  - Take a feature x_i
    - Replace it by (x_i - mean)/max
    - So your values all have an average of about 0
- Can also use **standard deviation for normalization (please apply this one!)**
  - Take a feature x_i
    - Replace it by (x_i - mean)/sd
    - So your values all have an average of about 0

Logistic Regression

- Where y is a discrete value
  - Develop the logistic regression algorithm to determine what class a new input should fall into
- Classification problems
  - Email -> spam/not spam?
  - Online transactions -> fraudulent?
  - Tumor -> Malignant/benign
- Variable in these problems is Y
  - Y is either 0 or 1
    - 0 = negative class (absence of something)
    - 1 = positive class (presence of something)
- Logistic regression is a **classification algorithm** - don't be confused

Logistic Regression
Hypothesis Representation

- What function is used to represent our hypothesis in classification
- We want our classifier to output values between 0 and 1
  - When using linear regression we did $h_\theta(x) = (\theta^T x)$
  - For classification hypothesis representation we do $h_\theta(x) = g((\theta^T x))$
    - Where we define g(z)
      - z is a real number
    - $g(z) = 1/(1 + e^{-z})$

Logistic Regression
Decision Boundary

- One way of using the sigmoid function is;
  When the probability of y being 1 is greater than 0.5 then we can predict y = 1
- Else we predict y = 0

Logistic Regression

Redefined Cost Function

- Linear Regression: cost($h_\theta(x^i)$, y) = 1/2($h_\theta(x^i)$ - y^i)^2
- We can also re-write this:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x), y)$$

    - This is the cost you want the learning algorithm to pay if the outcome is $h_\theta(x)$ and the actual outcome is y
    - If we use this function for logistic regression this is a **non-convex function** for parameter optimization
        - Could work....
- What do we mean by non convex?
    - We have some function - J(θ) - for determining the parameters
    - Our hypothesis function has a non-linearity (sigmoid function of $h_\theta(x)$)
        - This is a complicated non-linear function
    - If you take $h_\theta(x)$ and plug it into the Cost() function, and them plug the Cost() function into J(θ) and plot J(θ) we find many local optimum → *non convex function*
    - Why is this a problem
        - Lots of local minima mean gradient descent may not find the global optimum - may get stuck in a global minimum
    - We would like a convex function so if you run gradient descent you converge to a global minimum

Logistic Regression
What does this actually mean?

- To get around this we need a different, convex Cost() function which means we can apply gradient descent

$$\rightarrow J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \underline{\mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})}$$

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note: $y = 0$ or 1 always

OR

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

References

- https://www.coursera.org/learn/machine-learning/lecture/kCvQc/gradient-descent-for-linear-regression
- http://www.holehouse.org/mlclass/06_Logistic_Regression.html
- http://www.holehouse.org/mlclass/04_Linear_Regression_with_multiple_variables.html
- https://en.wikipedia.org/wiki/Gradient_descent