Lecture 3 [Logically Weighted & Logistic Regression]

Linear Regression(recap):

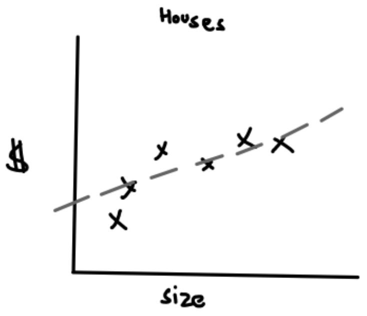
Recap: $(x^{(i)}, y^{(i)})$ — ith example

$x^{(i)} \in \mathbb{R}^{n+1}$, $y^{(i)} \in \mathbb{R}$, $x^{(0)} = 1$

m = # examples , n = # features

Hypothesis : $h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j = \theta^T x$ "Linear function of features x"

Cost Function: $J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$ "minimize as function of J to find the parameters Θ for your straight line fit to the data"



Feature selection algorithms are types of algorithms for automatically deciding what set of features does the best job fitting the data that you have if it's not fit well by a perfectly straight line

Logically weighted regression(depends on linear regression):

A way to modify linear regression to make it fit very non-linear functions (so you aren't just fitting straight lines)

In machine learning, we sometimes distinguish between **parametric learning algorithms** and **non-parametric learning algorithms**. In a parametric learning algorithm you fit some fixed set of parameters (Θ_i) to data

Locally weighted regression is a non-parametric learning algorithm. The amount of data/parameters you need to keep, grows (linearly in this case) with the size of the data/training set

With a parametric learning algorithms, no matter how big your training set is, you fit the parameters (Θ_i), then you could erase the training set from your computer memory and make predictions just using the parameters (Θ_i)

In a non-parametric learning algorithm, the amount of stuff you need to store around in computer memory or on disk, grows linearly as a function of the training set size

Locally weighted regression

Fit θ to minimize a modified cost function,

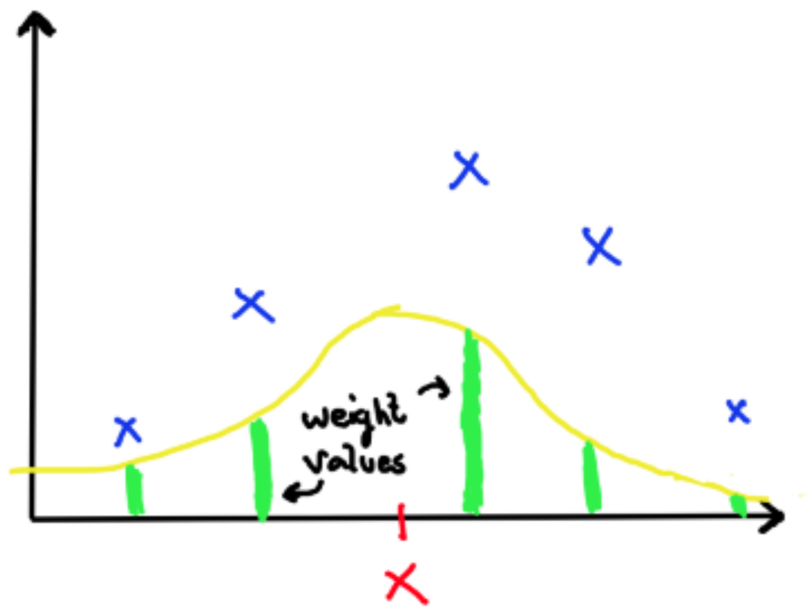
$$\sum_{i=1}^{m} w^{(i)} \left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

where $w^{(i)}$ is a "weighting" function

A common/default choice of $w^{(i)}$ is,

$$w^{(i)} = \exp\left(- \frac{(x^{(i)} - x)^2}{2}\right.$$

- If $|x^{(i)} - x|$ is small, $w^{(i)} \approx 1$
- If $|x^{(i)} - x|$ is large, $w^{(i)} \approx 0$

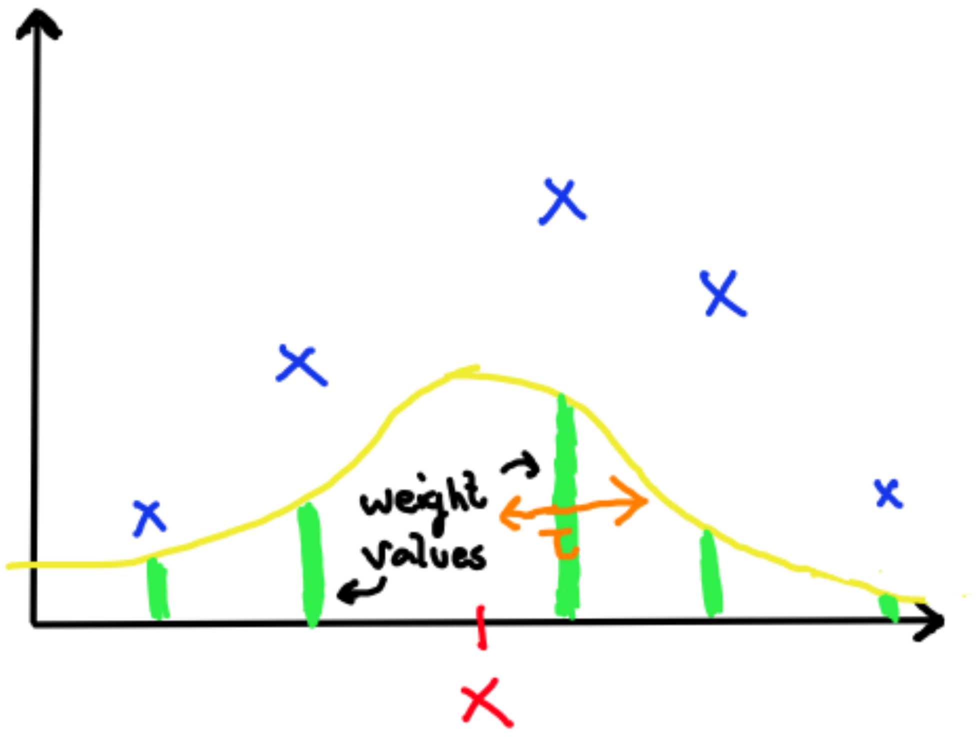


How do you choose the width Gaussian density? How fat/thin should it be? This decides how big a neighborhood should you look in order to decide what's the neighborhood of points you should use to fit this local straight line

A common/default choice of $w^{(i)}$ is,

$$w^{(i)} = \exp\left(-\frac{(x^{(i)}-x)^2}{2\tau^2}\right)$$

- If $|x^{(i)}-x|$ is small, $w^{(i)} \approx 1$
- If $|x^{(i)}-x|$ is large, $w^{(i)} \approx 0$



"Tau" : τ - "bandwith parameter"

This is a hyper-parameter of the algorithm

Depending on the choice of tau, you can choose a fatter or a thinner bell-shaped curve which causes you to look in bigger or a narrower window in order to decide how many nearby examples to use in order to fit the straight line

The choice of the bandwith tau has an effect on overfitting and underfitting. If tau is too broad, you end up over-smoothing the data and if tau is too thin, you end up fitting a very jagged fit to the data

Locally weighted linear regression is usually used when you have a relatively low dimensional dataset (when the number of features is not too big) and you have a lot of data and you don't want to think about what features to use

Probabilistic interpretation (of linear regression):

Why do we use least squares (squared error)?

Why Least squares?

Assume $$y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$$

"error": (unmodeled effects, random noise)

$$\varepsilon^{(i)} \sim \overbrace{\mathcal{N}(0, \sigma^2)}^{\text{normal/Gaussian distribution with mean } 0\text{, variance } \sigma^2}$$

"is distributed" "stripped n" "sigma"

$$P(\varepsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(\varepsilon^{(i)})^2}{2\sigma^2}\right)$$

Probability Density

- Assume the error terms are I.I.D. (Independently and Identically Distributed)

This implies that:

$$P(y^{(i)} \mid x^{(i)} ; \theta) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(\overset{\text{random variable}}{y^{(i)}} - \overset{\text{mean}}{\theta^T x^{(i)}})^2}{2\underset{\text{variance}}{\sigma^2}}\right)$$

"parameterized by"

I.e. $$y^{(i)} \mid x^{(i)} ; \theta \sim \mathcal{N}(\theta^T x^{(i)}, \sigma^2)$$

If you are willing to make those assumptions then linear regression falls out, almost naturally, of the assumptions you just made, and in particular, under the assumptions you just made:

$$\mathcal{L}(\Theta) = P(\vec{y} \mid x ; \Theta) = \prod_{i=1}^{m} P(y^{(i)} \mid x^{(i)} ; \Theta)$$

"Likelihood of Θ"

"probability of all values of y, $(y^1 \ldots y^m)$, given all x's and parameterized by Θ"

$$= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(y^{(i)} - \Theta^T x^{(i)})^2}{2\sigma^2}\right)$$

What's the difference between Likelihood and Probability?

The Likelihood of the parameters is exactly the same thing as the probability of the data

If you view the value of P (y | x ; Θ) as a function of the parameters holding the data fixed then we call that the Likelihood. If you think of the training set, the data is a fixed thing and then varying parameters Θ, then you're going to use the term Likelihood. Whereas if you view the parameters Θ as fixed and maybe varying the data, you're going to use the term Probability

"Likelihood of the parameters" === "Probability of the data"

"log likelihood"

$$\ell(\Theta) = \log \mathcal{L}(\Theta)$$

$$= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(y^{(i)} - \Theta^T x^{(i)})^2}{2\sigma^2}\right)$$

$$= \sum_{i=1}^{m} \left[\log \frac{1}{\sqrt{2\pi}\,\sigma} + \log \exp(\cdots)\right]$$

$$= m \log \frac{1}{\sqrt{2\pi}\,\sigma} + \sum_{i=1}^{m} \left[-\frac{(y^{(i)} - \Theta^T x^{(i)})^2}{2\sigma^2}\right.$$

• Log of a product is equal to the sum of the logs

"Maximum Likelihood Estimation" (MLE)

- Choose Θ to maximize $L(\Theta)$

  i.e. choose a value of Θ so that that value of Θ maximizes the probability of the data

To simplify the algebra, rather than maximizing the **L(Θ)** , it's actually easier to maximize the **l(Θ)**
The **log L(Θ)** is a strictly monotonically increasing function, so the value of Θ that maximizes the **l(Θ)**, should be the same as the value of Θ that maximizes the **L(Θ)**, and if you derive the **l(Θ)**

"Maximum Likelihood Estimation" (MLE)

- Choose θ to maximize $L(\theta)$

  i.e. choose a value of θ so that that value of θ maximizes the probability of the data

I.e. Choose θ to minimize $\frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)}-\theta^T x^{(i)}\right)^2 = J(\theta)$

Choosing the value of Θ to minimize the least squares errors is just finding the **MLE** for the parameters Θ under the set of assumptions you made such that the error terms are Gaussian and **IID**

Logistic regression(classification algorithm, probabilistic interpretation):

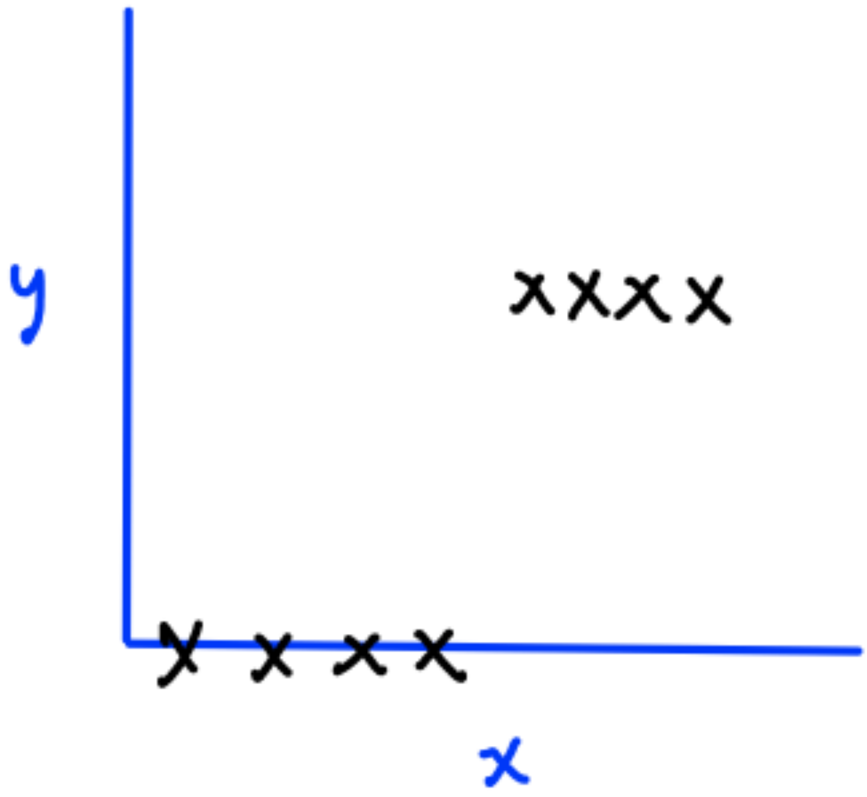
The key steps are:
1) Make an assumption about P of y given x, parameterized as Θ
2) Figure out maximum likelihood estimation

Take this framework and apply it to a different type of problem, where the value of y is now either 0 or 1

Classification

[Binary Classification]

$y \in \{0,1\}$



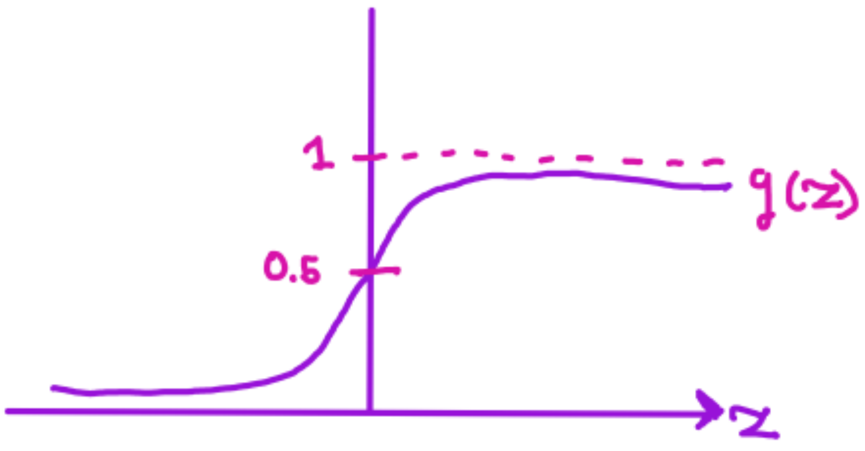
Something that's not a good idea is to apply Linear Regression to this data set. You know that for a clasification problem, the values are 0 or 1, so it outputs of negative values or value even greater seems strange

Logistic Regression

- You'll want $h_\theta(x) \in [0,1]$

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

$$g(z) = \frac{1}{1+e^{-z}}$$ "sigmoid" or "logistic" function



Let's make some assumptions about the distribution of y given x parameterized by Θ

Assume the data has the following distribution:

$$P(y=1 \mid x;\theta) = h_\theta(x)$$

Assume what you want your learning algorithm to do is input the features and tell you what's the chance that y is equal to 1

Assume the data has the following distribution:

$$\begin{cases} P(y=1 \mid x;\theta) = h_\theta(x) \\ P(y=0 \mid x;\theta) = 1 - h_\theta(x) \end{cases}$$

$$y \in \{0,1\}$$

Compress

$$P(y \mid x;\theta) = h_\theta(x)^{y} \left(1 - h_\theta(x)\right)^{1-y}$$

- If $y=1$, $(\text{anything})^0 = 1$, then $P(y \mid x;\theta) = h_\theta(x)$
- If $y=0$, then $P(y \mid x;\theta) = (1 - h_\theta(x))$

With maximum likelihood estimation, we'll want to find the value of Θ that maximizes the likelihood of the parameters

To make the algebra simpler, take the log of the likelihood and so compute the log likelihood

$$L(\theta) = P(\vec{y} \mid x; \theta)$$

$$= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta)$$

$$= \prod_{i=1}^{m} h_\theta(x^{(i)})^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}$$

"log likelihood"

$$l(\theta) = \log L(\theta)$$

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

- Choose the value of θ to try to maximize $l(\theta)$

- The algorithm we're going to use to choose θ to maximize $l(\theta)$, is Gradient ascent / Batch Gradient ascent

$$\underset{\text{update}}{\theta_j :=} \theta_j + \alpha \overset{\text{partial derivative}}{\frac{\partial}{\partial \theta_j}} l(\theta)$$

The last thing to really flesh out this algorithm is to plug in the definition of the hypothesis. If you do so, you will find that batch gradient ascent is the following:

$$\boxed{\text{Batch gradient ascent}}$$

$$\theta_j := \theta_j + \alpha \underbrace{\sum_{i=1}^{m} \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_j^{(i)}}_{\frac{\partial}{\partial \theta_j} l(\theta)}$$

Newton's method(algorithm for logistic regression):

Gradient ascent is a good algorithm, but it takes a lot of iterations(baby steps) for gradient ascent to converge. There's another algorithm called **Newton's method** which allows you to take much larger steps

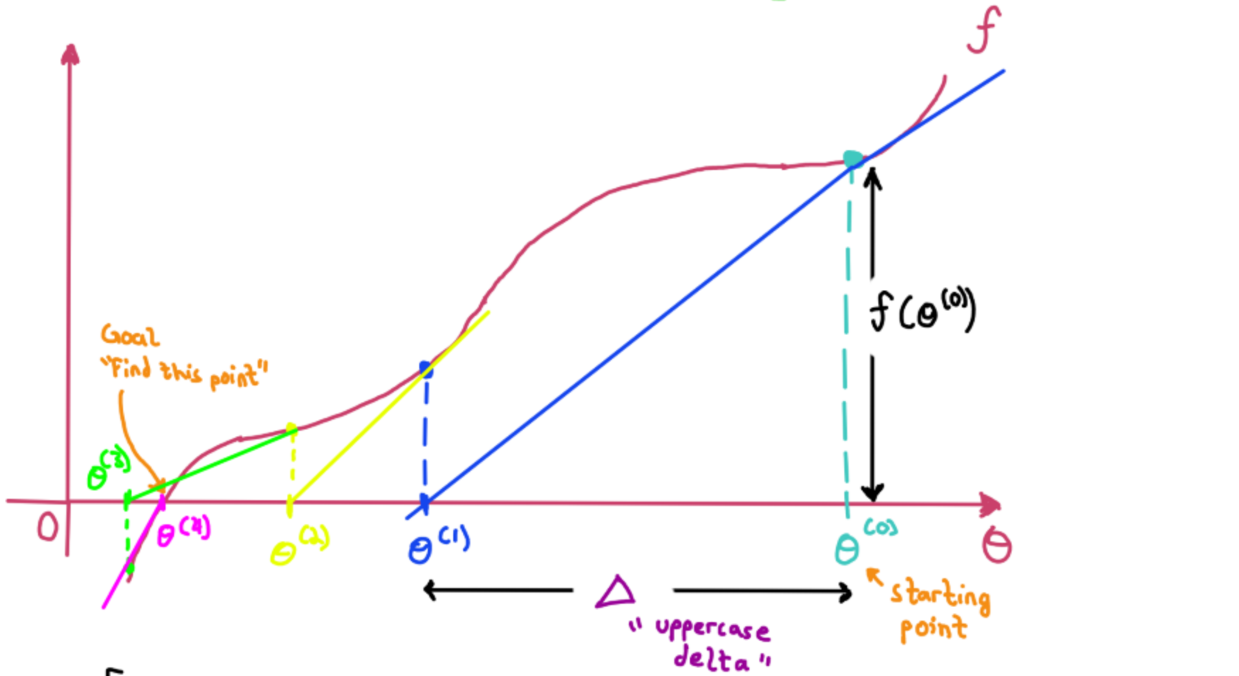
This algorithm is sometimes much faster than gradient ascent for optimizing the value of Θ

[Described with a simplified one-dimensional problem]

where Θ is a real number:

Newton's method

- Say you have some function f, and you want to find a θ s.t. $f(\theta)=0$
  This is a problem that Newton's method solves
- What you really want is to maximize $l(\theta)$ (at maximum, first derivative must be 0)
  [i.e. you want to value where the derivative $l'(\theta)=0$]



[this is how 1 iteration of Newton's method will work]

δ " lowercase delta "

What we'd like to do is solve for the value of delta because one iteration of Newton's method is:

$$\theta^{(1)} := \theta^{(0)} - \Delta$$

$$f'(\theta^{(0)}) = \frac{f(\theta^{(0)})}{\Delta}$$

$$\Delta = \frac{f(\theta^{(0)})}{f'(\theta^{(0)})}$$

$$\theta^{(t+1)} := \theta^{(t)} - \frac{f(\theta^{(t)})}{f'(\theta^{(t)})}$$

Let $f(\theta) = l'(\theta)$ ← we want to find the place where first derivate of l is 0

$$\theta^{(t+1)} := \theta^{(t)} - \frac{l'(\theta^{(t)})}{l''(\theta^{(t)})}$$

Newton's method is a very fast algorithm and enjoys a property called **Quadratic convergence**. What it means is, after one iteration Newton's method has 0.01 error (0.01 away on x-axis from the true value of f = 0), the error could go to 0.0001 error. After 2 iterations it could go to 0.0000001 error

Newton's method, under certain assumptions that functions move not too far from quadratic, the number of significant digits that you have convergedto the minimum, doubles on a single iteration. So this is called **quadratic convergence**

When you get near the minimum, Newton's method converges extremely rapidly, which is why Newton's method requires relatively few iterations

When θ is a vector: $(\theta \in \mathbb{R}^{n+1})$

$$\theta^{(t+1)} := \theta^{(t)} + \overbrace{H^{-1}}^{\mathbb{R}^{n+1 \times n+1}} \underbrace{\nabla_\theta \ell} \quad \text{vector of derivatives } (\mathbb{R}^{n+1})$$

where H is the Hessian matrix

$$H_{ij} = \frac{\partial^2 \ell}{\partial \theta_i \, \partial \theta_j} \quad \leftarrow \text{defined as the matrix of partial derivatives}$$

The disadvantage of Newton's method is that in **high-dimensional problems**, if Θ is a vector, then each step of Newton's method is much more expensive because you're either solving linear system equations or having to invert a pretty big matrix

RULES OF THUMB:

If the number of parameters is not too big (10, 50, etc) so that the computational cost per iteration is manageable, you would likely use Newton's method because it converges in a very small number of iterations and can be a much faster algorithm than gradient descent, but if you have a very large number of parameters (10,000, 50,000, etc), you would use gradient descent instead