https://www.overleaf.com/5851353818ykkkryygfhnh

Regressions and Gradient Descent

Machine Learning Decal

Hosted by Machine Learning at Berkeley

Agenda

Linear Regression?

Optimization via Gradient Descent

5-minute break: http://tinyurl.com/mldecallec2

Logistic Regression

Multinomial Regression

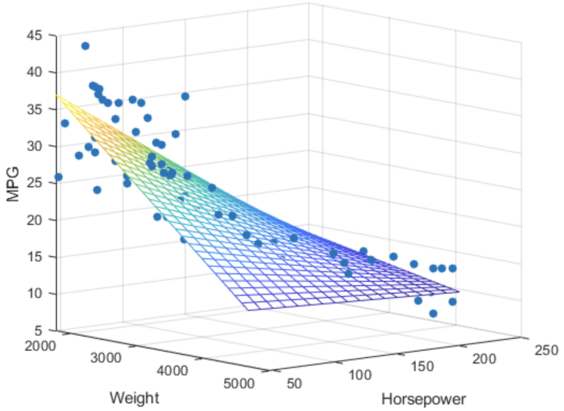
Questions

Extras

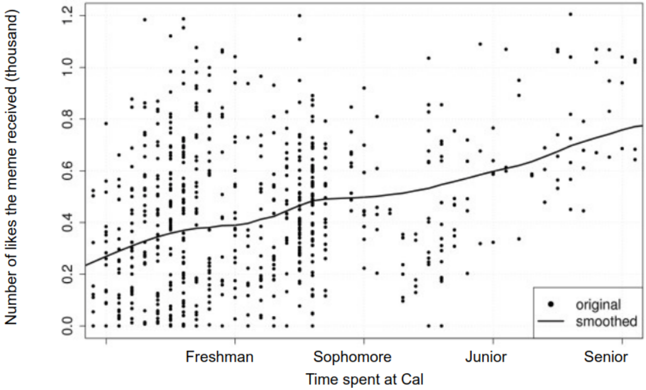
Linear Regression?

Linear Regression?

Important approach for relationships between continuous variables: Finding linear relationships



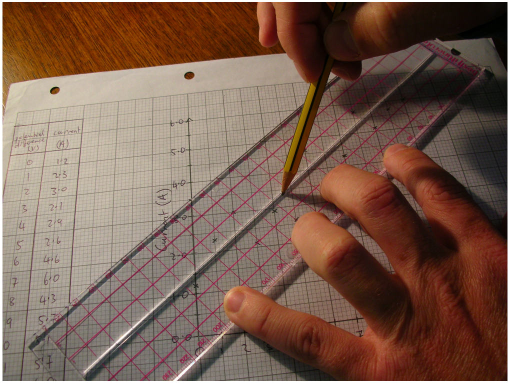
Dankness of meme vs time spent at Cal



Question: Given any arbitrary time a student spent at Cal, can you tell me how dank the meme he/she creates will be?

Best Fit Line

Easy. You take out a



and squint real hard to draw a best fit line.

JK

This is a bad idea... we have computers... and ML!

Lets formalize this method with math

To draw a straight line, you need 2 pieces of information:

- the slope, b_0
- the y-intercept, b_1

So you have an expression of the straight line $h(x)$ that you are trying to draw:

$$h(x) = b_0 + b_1 x$$

Now the question becomes:

Given (x_i, y_i) pairs, how do you find b_0 and b_1 that give you the best-fit line?

What is best-fit?

Our expression:

$$h(x) = b_0 + b_1 x$$

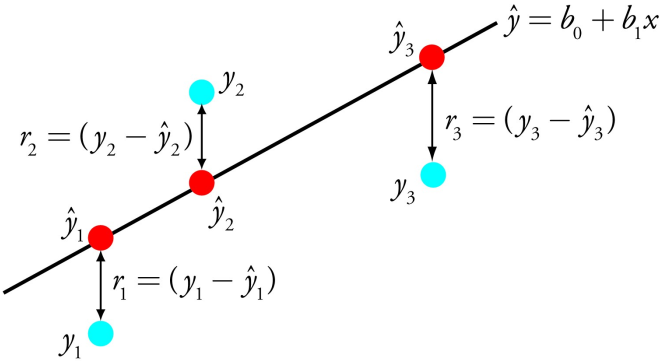
Can we define an error metric such that we make the most accurate predictions when minimizing it?

We argue that mean-squared error is the best way to approach this problem.

Minimizing Error

Let $\hat{y}_i = h(x) = b_0 + b_1 x$

$$\min J(b_0, b_1)$$

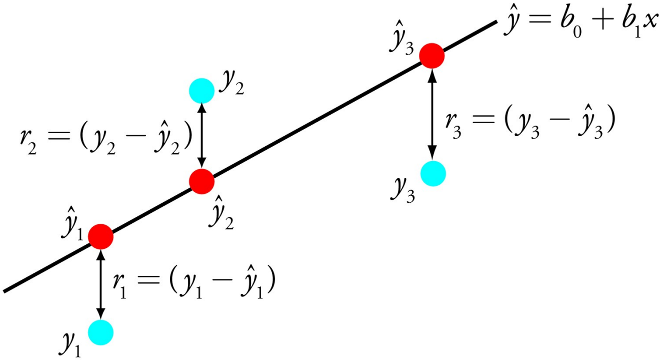


$\hat{y} = b_0 + b_1 x$
$\hat{y}_3$
y_2
$r_3 = (y_3 - \hat{y}_3)$
$r_2 = (y_2 - \hat{y}_2)$
y_3
$\hat{y}_2$
$\hat{y}_1$
$r_1 = (y_1 - \hat{y}_1)$
y_1

Minimizing Mean Squared Error

Let $\hat{y}_i = h(x) = b_0 + b_1 x$

$$\min J(b_0, b_1) = \frac{1}{2m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

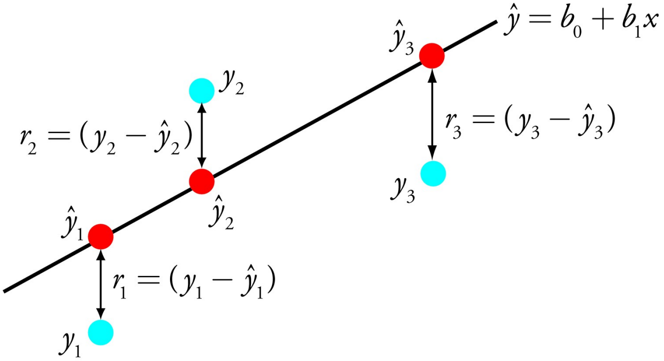


$\hat{y} = b_0 + b_1 x$
$\hat{y}_3$
y_2
$r_2 = (y_2 - \hat{y}_2)$
$r_3 = (y_3 - \hat{y}_3)$
y_3
$\hat{y}_2$
$\hat{y}_1$
$r_1 = (y_1 - \hat{y}_1)$
y_1

Minimizing Mean Squared Error

Let $\hat{y}_i = h(x_i) = b_0 + b_1 x_i$

$$\min J(b_0, b_1) = \frac{1}{2m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (y_i - b_0 - b_1 x_i)^2$$



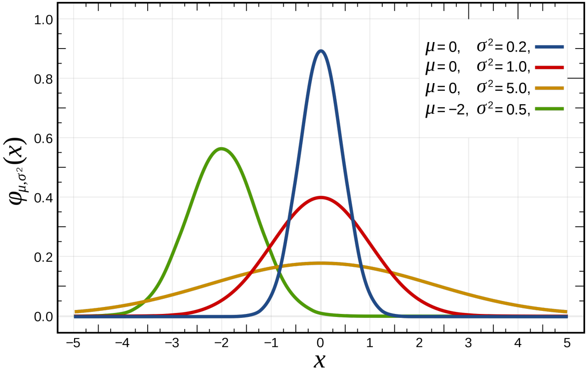
$\hat{y} = b_0 + b_1 x$
$\hat{y}_3$
y_2
$r_2 = (y_2 - \hat{y}_2)$
$r_3 = (y_3 - \hat{y}_3)$
y_3
$\hat{y}_2$
$\hat{y}_1$
$r_1 = (y_1 - \hat{y}_1)$
y_1

Why mean SQUARED?

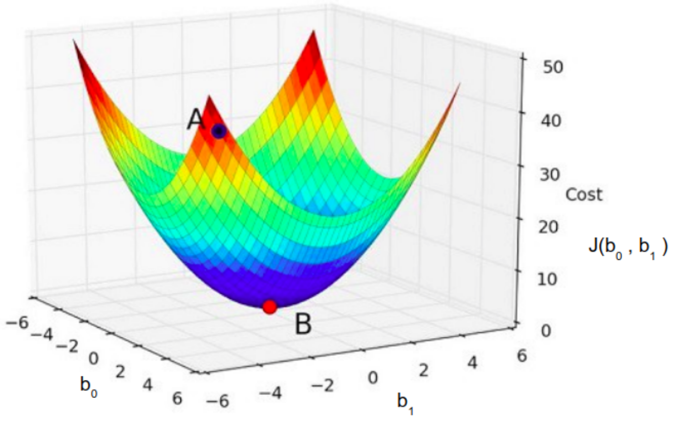
- Absolute values of error also fine, called L1 loss
- But what about random noise in our samples?
- Central limit theorem: noise is distributed via the normal distribution
- Best b_0, b_1: tightest distribution of error around the predicted value– least variance
- Minimizing the the least squared error is the same thing as minimizing the variance (s^2)!.

$$s^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Why mean SQUARED?



Visualizing the Cost Function



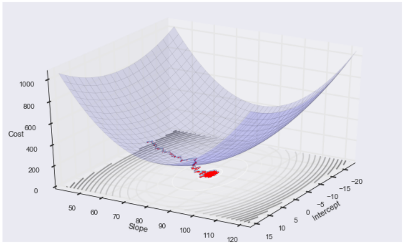
Get point $B = (b_0, b_1)$ such that $J(b_0, b_1)$ is the smallest.

Optimization via Gradient Descent

Minimizing the cost function

We can use a search algorithm that follows the scheme:

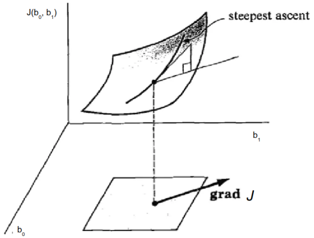
- Choose an initial guess for b_0, b_1
- Repeatedly update b_0, b_1 to make $J(b_0, b_1)$ smaller
- Keep doing this until $J(b_0, b_1)$ reaches its minimum



Gradient

Let's say $J(b_0, b_1) = 3{b_0}^2 b_1$

$$\textbf{grad}\, J = \langle \frac{\partial J}{\partial b_0}, \frac{\partial J}{\partial b_1} \rangle = \langle 6 b_0 b_1, 3{b_0}^2 \rangle$$



grad J is the vector that points in the direction with the largest increase (steepest ascent)

Equations for Gradient Descent

Cost Function:

$$J(b_0, b_1) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2$$

Objective:

$$\min_{b_0, b_1} J(b_0, b_1)$$

Derivatives (to determine the direction of descent):

$$\frac{\partial}{\partial b_0} J(b_0, b_1) = \frac{1}{m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})$$

$$\frac{\partial}{\partial b_1} J(b_0, b_1) = \frac{1}{m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

Gradient Descent Algorithm

1. Initialize random b_0, b_1
2. Repeat until convergence $\{$

$$b_0 := b_0 - \alpha \frac{\partial}{\partial b_0} J(b_0, b_1)$$

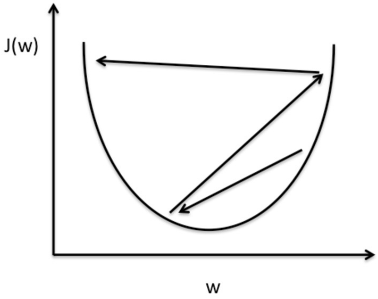
$$b_1 := b_1 - \alpha \frac{\partial}{\partial b_1} J(b_0, b_1)$$

$\}$

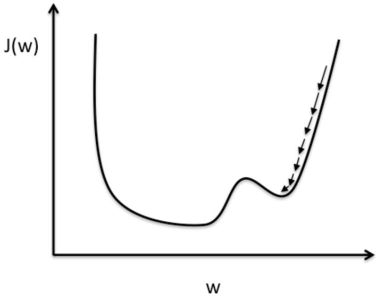
$$\frac{\partial}{\partial b_0} J(b_0, b_1) = \frac{1}{m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})$$

$$\frac{\partial}{\partial b_1} J(b_0, b_1) = \frac{1}{m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

Choosing the learning rate α



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

The best fit straight line

We have just computed b_0, b_1 that gives us the best fit straight line

$$h(x) = b_0 + b_1 x$$

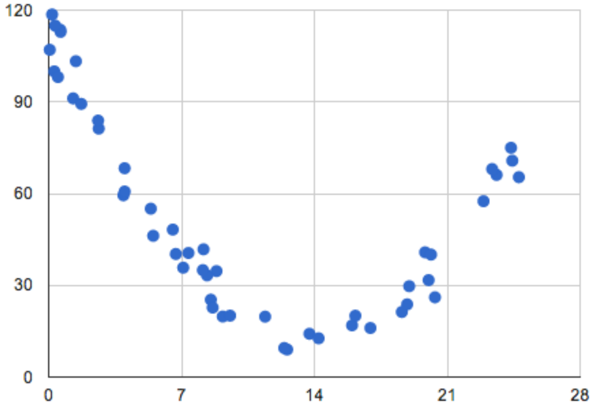
to our dataset (x_i, y_i)!
So now given any arbitrary value x, we have a model $h(x)$ that can predict what the corresponding best prediction y will be.

Questions:

- But is a straight line always the line of best fit?
- Can the cost function $J(b)$ be smaller?

Nonlinear relationships

What if you are given this dataset instead?



Linearization

The best model is now a quadratic expression:

$$h(x) = b_0 + b_1 x + b_2 x^2$$

Can be linearized as:

$$h(x_1, x_2) = b_0 + b_1 x_1 + b_2 x_2$$

where $x_1 = x$, $x_2 = x^2$
This method can be generalized to any polynomials!

$$h(x) = b_0 + b_1 x_1 + b_2 x_2^2 + \ldots + b_n x_n^n$$

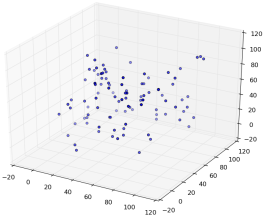
Multidimensional input

We can run linear regression in arbitrary dimensions (x_1, x_2, y)

$x_1 =$ The time spent at Cal

$x_2 =$ The time spent on Facebook

$y =$ The dankness of memes



$$h(x_1, x_2) = b_0 + b_1 x_1 + b_2 x_2$$

Try visualizing the cost function, $J(b_0, b_1, b_2)$

Impossible! Beyond 3-dimensional, we need Linear Algebra

Matrix form

$x_j^{(i)}$: i^{th} sample, j^{th} feature

$$h(x_1^{(i)}, x_2^{(i)}) = b_0 + b_1 x_1^{(i)} + b_2 x_2^{(i)}$$

Let $\hat{y}^{(i)} = h(x_1^{(i)}, x_2^{(i)})$

$$\begin{pmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{pmatrix} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}$$

$$\vec{\hat{y}} = \mathbf{X}\vec{b}$$

Generalize to n dimensions in terms of $\vec{y}, \mathbf{X}$ and $\vec{b}$

Let $\vec{e} = \vec{y} - \vec{\hat{y}}$

$$\vec{y} = \vec{\hat{y}} + \vec{e}$$

$$\vec{y} = \mathbf{X}\vec{b} + \vec{e}$$

$$\begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

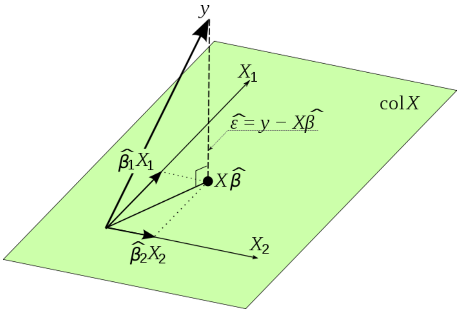
How do we compute $\vec{b}$

Want to minimize $\vec{e}$ for $\vec{y} = \mathbf{X}\vec{b} + \vec{e}$

Can approximate using Least Squares Method:

$$\mathbf{X}^T\vec{y} = \mathbf{X}^T\mathbf{X}\vec{b}$$

$$\vec{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y}$$



5-minute break:
http://tinyurl.com/mldecallec2

Logistic Regression

Continuous vs Discrete Output

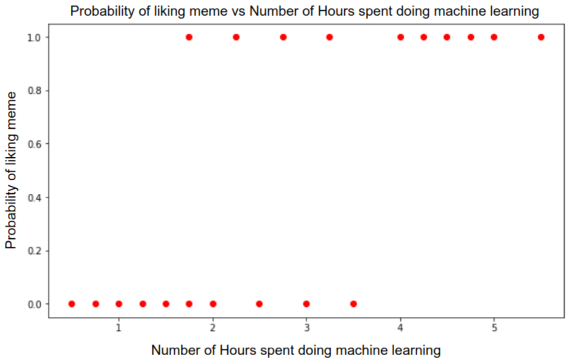
As an active member of the Cal community, you would like to know whether specific groups like your memes.

Deep Learning
What society thinks I do
What my friends think I do
What other computer scientists think I do
What mathematicians think I do
What I think I do
from theano import *
What I actually do

Given this sadly relatable meme and the time an individual spent doing machine learning, predict whether he/she will like this meme

Fit This

Why is linear regression a sub-optimal algorithm for this problem?



A Different Sort of Regression

- Recall that linear regression is for **continuous** dependent variables, e.g. Dankness of memes vs. Time spent at Cal.
- The current problem has **categorical** values for the dependent variable, i.e. Like/No like - only two **classes**
- We are regressing on (the likelihood of) **membership to a class**.

The Logistic Function

We introduce the logistic function, also called the **sigmoid**:

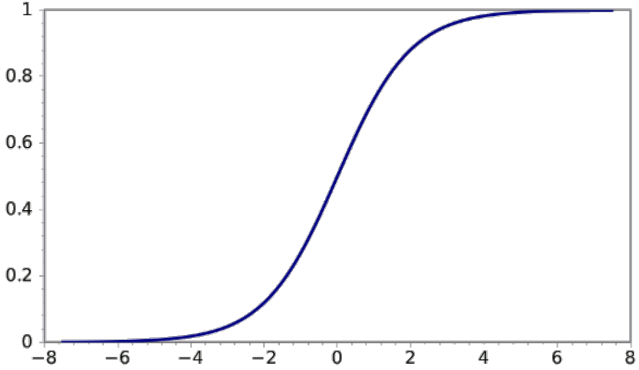
$$s(x) = \frac{1}{1 + e^{-x}}$$

- What is its domain and range?
- What is an interesting property regarding $s(x = 0)$?

Logistic Function: Dissected

Here's a picture. Notice that $s(x) - \frac{1}{2}$ intuitively appears to be an odd function. Hence, we have the following property for all x:

$$s(x) + s(-x) = 1$$



Why?

Recall from Linear Regression, that our hypothesis has the form:

$$h(x_1, x_2, \ldots, x_n) = b_0 + b_1 x_1 + \ldots + b_n x_n = \vec{b}^T \vec{x}$$

And, in our new problems, our dependent variables y are in $\{0, 1\}$.

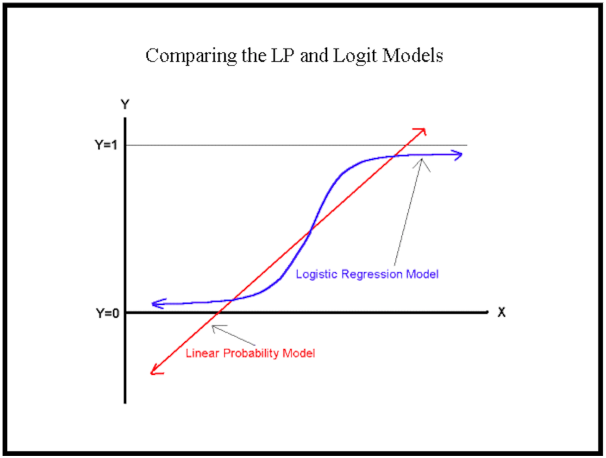
So, we propose that the hypothesis for Logistic Regression be:

$$h(\vec{x}) = s(\vec{b}^T \vec{x}) = \frac{1}{1 + e^{-\vec{b}^T \vec{x}}}$$

Sanity Check

- The range of $h(x)$ is $(0, 1)$. This is good – we round in order to classify.
- $h(x) + h(-x) = 1 \ \forall x$. Our model dictates that the probability of membership to one of the two classes is 1. This is good too.
- Finally, we note that $h(x)$ is also known as an **expit**. It converts log-probability into a probability. The inverse function is known as a **logit**.

The Role of Sigmoid



The Mission

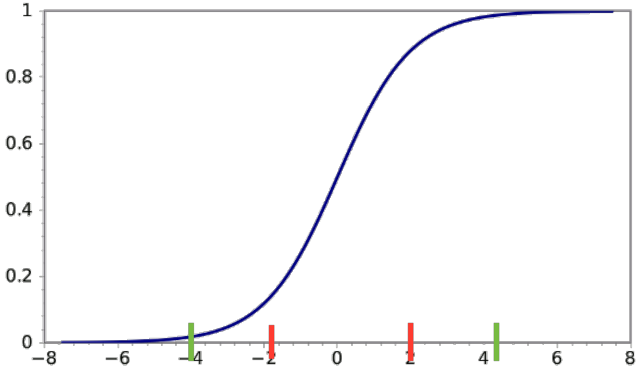
Our hypothesis is still parameterized by $b : [b_0 ... b_n]$.
The ideal b would be such that for all x such that $y = 1$, we have

$$h(x) \approx 1 \rightarrow \vec{b}^T \vec{x} >> 0$$

A similar conclusion follows for $y = 0$.

The Mission: Visualized

Where would **ideal** values of $\vec{b}^T\vec{x}$ fall on the x-axis relative to the colored bands?



The Logistic Loss Function

To improve our weights, we will seek to minimize the Logistic Loss Function, with $z = h(x) = \frac{1}{1+e^{-\vec{b}^T \vec{x}}}$:

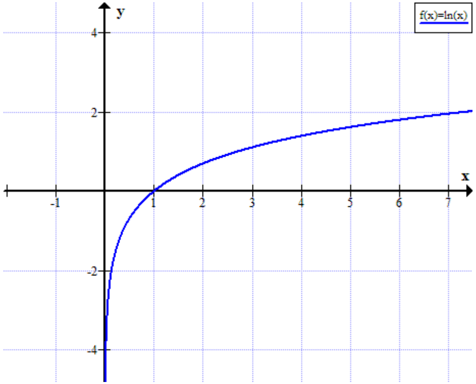
$$J(b) = -\sum_{i=1}^{m} \left(y^{(i)} \cdot \ln z^{(i)} + (1 - y^{(i)}) \cdot \ln (1 - z^{(i)}) \right)$$

Because $\forall i, y_i \in \{0, 1\}$, only one of the addends is nonzero. That addend will be minimized when the expression involving z_i is closest to zero. (Remember $\ln(1) = 0$).

The Natural Log

$z = h(x) = \frac{1}{1+e^{-\vec{b}^T\vec{x}}}$:

$$J(b) = -\sum_{i=1}^{m}\left(y^{(i)} \cdot \ln z^{(i)} + (1 - y^{(i)}) \cdot \ln\,(1 - z^{(i)})\right)$$



How to minimize $J(b)$

Again, we need to find the partial derivative with respect to b

$$\frac{\partial J(b)}{\partial b} = \sum_{i=1}^{m} -\frac{\partial z^{(i)}}{\partial b} \cdot \frac{\partial}{\partial z^{(i)}} J(b)$$

(see appendix for details)

$$\frac{\partial J(b)}{\partial b} = -\sum_{i=1}^{n} (y^{(i)} - z^{(i)}) X_i$$

Expressed in Matrix form:

$$\frac{\partial J(\vec{b})}{\partial \vec{b}} = -X^T(\vec{y} - s(X\vec{b}))$$

Gradient Descent Update

The (mini-)batch gradient descent update rule then becomes:

$$b \leftarrow b + \alpha X^T(y^{(i)} - s(Xb))$$

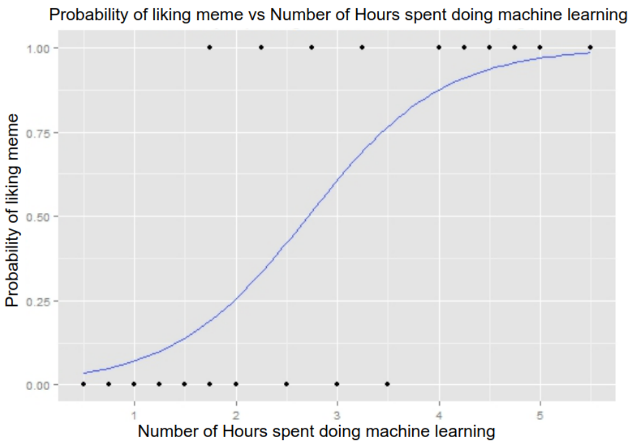
For stochastic gradient descent, the update is:

$$b \leftarrow b + \alpha(y^{(i)} - s(X_i b))X_i$$

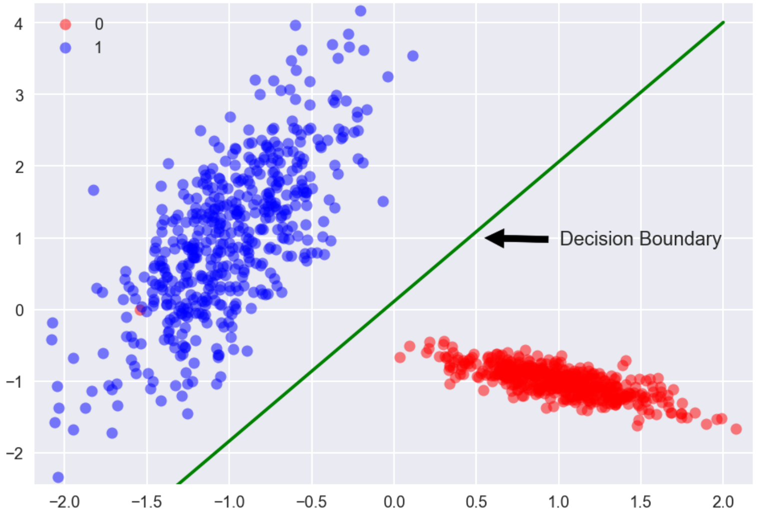
Batch gradient descent: whole dataset Stochastic gradient descent: uses a single datapoint

The End Result

Just for reference, here is the logistic regression solution to the initial problem that was posed:



Reminder: Linear Decision Boundary



Conceptual Summary: Linear vs. Logistic

Here we have some of the key distinctions between the two kinds of regression:

	Linear	Logistic
Label Type	Continuous	Categorical
Problem Type	Actual Regression	Actually Classification
Hypothesis	$\theta^T x$	$s(\theta^T x)$
Loss	Mean Squared	Logistic
Analytical Solution	Yes	No

Multinomial Regression

Multiple Choice

Instead of predicting the number of likes the meme gets, you want to predict the number of different reactions the meme gets.

If logistic regression answers True/False, then multinomial regression answers multiple choice.
The likelihood of any of the choices being correct forms a probability distribution.

Parameterizing Multinomial Regression

Earlier, having one vector $\vec{b}$ was sufficient.

- If we have possible classes: $c_1, ..., c_k$
- Then we will need parameter vectors $\vec{b_1}, ...\vec{b_k}$.
- We can store these in parameter matrix: $\mathbf{B} \in \mathcal{R}^{k \times d}$, with each parameter vector being a row $\vec{b_i}$

Calculating Actual Probability

We can give each class c_i a likelihood score:

$$\vec{b}_i \vec{x} \approx P(y = c_i)$$

The product of the matrix **B** and vector x results in a vector z which needs to be **normalized.** We used sigmoid before.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

The Softmax Function

We use the **softmax function** $S(z_i)$ to normalize multinomial regression scores:

$$P(y = c_i) = S(z_i) = \frac{e^{b_i x}}{\sum_{j=1}^{k} e^{b_j x}}$$

Note the denominator is the sum of all the exponentiated scores in the vector z.

Softmax Loss and Gradient Update

The softmax loss function $J(b)$ is actually just a generalization of the logistic loss function:

$$J(b) = \sum_{i=1}^{m} \sum_{j=1}^{k} 1\{y^{(i)} = c_j\} \ln\left(z_j\right)$$

Regularization: Prevents overfitting

L1: Useful for data with sparse features

L2: Useful when potential for a lot of noise in data

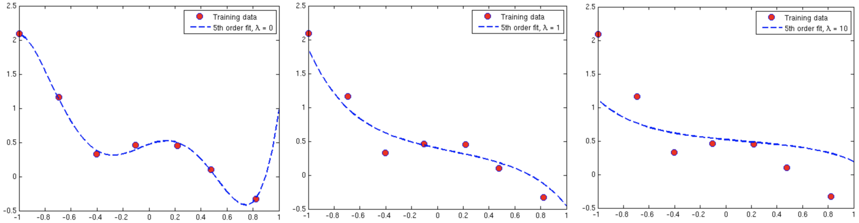
L1 regularization on least squares:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \boxed{\lambda \sum_{i=1}^{k} |w_i|}$$

L2 regularization on least squares:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \boxed{\lambda \sum_{i=1}^{k} w_i^2}$$

Different L2 regularization parameters



Modifying the Analytical Solution to Least Squares

In the first half of the lecture, we covered the normal equations solution to least squares regression. Now, we will do so for the regularized version:

$$\hat{b} = argmin_b ||y - Xb||_2^2 + \lambda ||b||_2^2$$

$$= argmin_b y^T y - 2b^T X^T y + b^T X^T X b + \lambda b^T b$$

Taking the derivative and setting equal to zero is allowed since the function is convex in b

$$-2X^T y + 2X^T X\hat{b} + 2\lambda\hat{b} = 0$$

$$(X^T X + \lambda I_d)\hat{b} = X^T y$$

$$\hat{b} = (X^T X + \lambda I_d)^{-1} X^T y$$

Lecture Recap

- Learned linear and logistic regression (and multinomial)
- Used linear algebra to derive a closed form solution
- Used gradient descent to iteratively optimize to a solution

Questions

Questions?

Extras

Probability Perspective

- Regression is a good summary of data, assuming the data has some key properties
- We need to know what those assumptions are, where they come from, and what to do when they fall apart

Assumptions: what are they?

- Linearity
- Normality of errors

$$\epsilon_i \sim N(0, \sigma^2)$$

- Homoscedasticity (constant variance)

$$Var(\epsilon_i) = Var(\epsilon_j)$$

- Independence of errors

$$\epsilon_i \epsilon_j \qquad \forall i \neq j$$

Data Generation Process

- The real world has natural processes. These processes can be collected/observed as data
- We try to build a model that mimics the real world model as close as possible
- Problem: there are some factors we can directly observe, and others that we can't
- Problem: we don't know what model the world uses
- In order to reason this process more rigorously, we can construct a "toy universe" to analyze our models

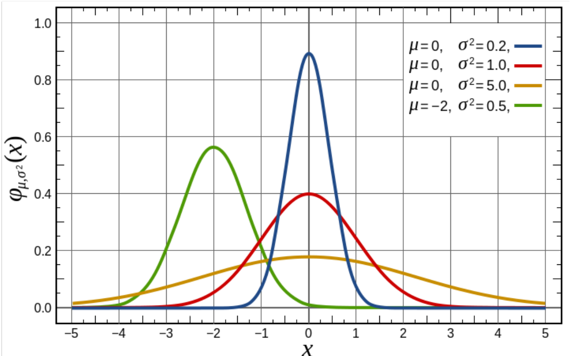
Data Generation Process

In our toy universe, data is generated through a linear model:

$$Y = X\theta$$

We can observe these Y values, but our observations have some noise in them. So our collected data looks like this:

$$Y = X\theta + Z \text{ where } Z \sim \mathcal{N}(0, \sigma^2)$$



Data Generation Process

So we have our noisy data:

$$Y = X\theta + Z$$

- We want to find the θ that can best replicate the data we've already observed
- Aka we want to **increase the likelihood** of the observed data being generated by the model

$$\hat{\theta} = argmax_{\theta} P(y_1, y_2, ..., y_n | \theta, x_1, x_2, ..., x_n)$$

Maximum Likelihood

$$\hat{\theta} = argmax_\theta P(y_1, y_2, ..., y_n | \theta, x_1, x_2, ..., x_n)$$

$$\hat{\theta} = argmax_\theta \prod_i P(y_i|\theta, x_i) = argmax_\theta \sum_i \log P(y_i|\theta, x_i)$$

Remember that $y_i \sim \mathcal{N}(\theta^T x_i, \sigma^2)$:

$$\hat{\theta} = argmax_\theta \sum_i \log \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y_i - \theta x_i)^2}{2\sigma^2}}$$

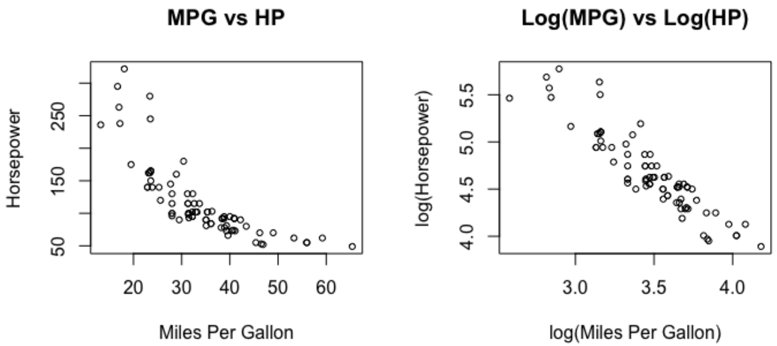
$$\hat{\theta} = argmax_\theta \sum_i \log e^{-\frac{(y_i - \theta x_i)^2}{2\sigma^2}} = argmax_\theta \sum_i -\frac{(y_i - \theta x_i)^2}{2\sigma^2}$$

$$\boxed{\hat{\theta} = argmin_\theta \sum_i (y_i - \theta^T x_i)^2}$$

Assumptions: what do we do if they are not satisfied?

- If the data is nonlinear...
    - Try performing a transformation on the independent or dependent variables such as squaring it, taking the log or square root, or ...
- If the errors are not normal...
    - Often, this isn't a big problem
    - Transformations help here too
    - Maybe subsets of the data are more normal than the overall set
    - Outliers and/or high leverage points may contribute to this issue

Example of the beauty of a log transform



Derivation of $\frac{\partial J(b)}{\partial b}$

First, we arrange all the data points into a **design matrix**, X, where point $x^{(i)}$ is the i^{th} row: X_i. We then find the appropriate gradient to solve $\min J(b)$.

$$\frac{\partial J(b)}{\partial b} = \sum_{i=1}^{m} - \frac{\partial z^{(i)}}{\partial b} \cdot \frac{\partial}{\partial z^{(i)}} J(b)$$

The first half of the chain rule:

$$\frac{\partial z^{(i)}}{\partial b} = \frac{\partial}{\partial b} s(b^T X_i) = s(b^T X_i)(1 - s(b^T X_i))X_i = z^{(i)}(1 - z^{(i)})X_i$$

You may want to check for yourself that $s'(x) = s(x)(1 - s(x))$

Derivation of $\frac{\partial J(b)}{\partial b}$

The other half of the chain rule:

$$\frac{\partial}{\partial z^{(i)}} J(b) = \frac{\partial}{\partial z^{(i)}} (y^{(i)} \cdot \ln z^{(i)} + (1 - y^{(i)}) \cdot \ln (1 - z^{(i)}))$$

$$= \frac{y^{(i)}}{z^{(i)}} - \frac{1 - y^{(i)}}{1 - z^{(i)}}$$

Hence, we have:

$$\frac{\partial J(b)}{\partial b} = -\sum_{i=1}^{n} \left(\frac{y^{(i)}}{z^{(i)}} - \frac{1 - y^{(i)}}{1 - z^{(i)}} \right) z^{(i)}(1 - z^{(i)}) X_i = -\sum_{i=1}^{n} (y^{(i)} - z^{(i)}) X_i$$

Expressed in Matrix form:

$$\frac{\partial J(\theta)}{\partial \theta} = -X^T (y - s(X\theta))$$