Lecture 5: Regression, Part 1
Modeling Social Data, Spring 2017
Columbia University

February 24, 2017

Notes from ha2434

1 Regression Task

We have a set of predictors (also called inputs or features) $\{x_i\}_{i=1}^N$ with their corresponding outcomes $\{y_i\}_{i=1}^N$ where each x_i is a k dimensional vector. We want to learn a function f from the data such that

$$\hat{y} = f(x) \tag{1}$$

where $\hat{y}$ is our prediction.
To evaluate how well f predicts, we define a loss function.

2 Loss Function

The squared loss $L[f]$ is given by

$$L_i[f] = (y_i - \hat{y}_i)^2 \tag{2}$$

where $\hat{y}_i = f(x_i)$ Then the total loss over all the N data vectors is

$$L[f] = \frac{1}{N}\sum_{i=1}^{N} L_i[f] = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2 \tag{3}$$

It may seem a bit abstract why we are using this particular loss function and where it came from. To study the motivation behind this, we define the Maximum Likelihood.

3 Maximum Likelihood Estimation

There are two steps to define the MLE:

- Assume some family of probabilistic models generated from the data.
- Find the model under which the observed data is most likely.

This can be mathematically framed as

$$f^* = \arg\max_{x} p(D|f) \tag{4}$$

A common assumption is

$$y_i = f(x_i) + \epsilon_i \tag{5}$$

where, $\epsilon_i = N(0, \sigma^2)$ is normally distributed additive noise. Thus,

$$P(\epsilon_i|f) = P(y_i - f(x_i)|f) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{1}{2\sigma^2}(y_i - f(x_i))^2) \tag{6}$$

The likelihood is then given by

$$P(D|f) = \prod_{i=1}^{N} P(\epsilon_i|f) = (2\pi\sigma^2)^{-N/2} exp(-\frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - f(x_i))^2) \tag{7}$$

Noticing that the exponents form a sum, we can instead take the log likelhood to simplify the expression. Moreover, since the log function is a monotonically increasing function, this will not affect the optimal solution.

$$\log P(D|f) = -\frac{N}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - f(x_i))^2 \tag{8}$$

A closer inspection tells us that the first term is independent of f and the second term is nothing but a scaled version of the loss function $L[f]$ as defined earlier. Therefore, we can conclude that minimizing the squared loss is equivalent to maximizing the (log) likelihood assuming additive Gaussian noise.

4 Finding f(x)

Note on matrix notation:

- X is a Nxk matrix where each column X_i is the i^{th} feature of all data points $x_1 \cdots x_N$ and the first column X_1 corresponds to the intercept and is equal to 1
- w is a kx1 column matrix where the first element corresponds to the bias.
- y is a Nx1 column matrix where y_i is the predictor for the i^{th} data point x_i

To find f, we assume a linear model in the weight vector w

$$\hat{y} = f(x) = Xw \tag{9}$$

For a single data point, this can be written as

$$\hat{y_i} = f(x_i) = x_i^T w = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_k x_{ik} \tag{10}$$

Now the task of finding f boils down to finding k numbers $w_1 \cdots w_k$. This is known as a parametric model.

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} (y_i - wx_i)^2 \tag{11}$$

This can be done in any one of the following ways each of which has its own limitations and use-cases.

4.1 Brute Force

This is the most naive method which is generally not feasible in many practical problems. In this, all possible values of w are tried and the one at which the loss function is minimum is chosen to be the optimal w^*. As the dimension of the weight vector increases, the number of possible choices of w also increases exponentially, and this can become a near-impossible task.

4.2 Normal Equations

In this we try to minimize the loss function by equating its gradient w.r.t w to zero.

$$\frac{\partial L}{\partial w} = \frac{1}{N} \sum \frac{\partial}{\partial w} (y_i - wx_i)^2 = 0 \tag{12}$$

Solving this, we get

$$\frac{\partial L}{\partial w} = -\frac{2}{N} \sum_{i=1}^{N} (y_i - wx_i)x_i = 0 \tag{13}$$

In matrix notation, this can be written as

$$X^T(y - Xw) = 0 \tag{14}$$

This gives,

$$\hat{w} = (X^T X)^{-1} X^T y \tag{15}$$

This is called Normal Equation.

4.2.1 Cost

Analyzing the cost in terms of running time we get:

- $X^T y = O(kN)$
- $X^T X = O(k^2 N)$
- $(X^T X)^{-1} = O(k^3)$

Thus, the total cost is $O(k^2 N + k^3)$. This is good if k is small (of the order 100). However, when k is very large this will take too much time to compute. Thus, this method is not suitable for very wide data ($k >> N$).

4.3 Batch Gradient Descent

Instead of directly computing w^* which corresponds to the w at the minima of the loss function, we can take an iterative approach. This is based on the idea that if we start from some (random) point and follow the loss surface, we will eventually get to the bottom of it. Since gradient gives the direction of steepest ascent, we follow the direction of the negative of gradient which gives the direction of steepest descent. Thus, the updation step to calculate the new weight vector becomes:

$$w = w - \eta \frac{\partial L}{\partial w} \tag{16}$$

where, w is the current weight vector, $\frac{\partial L}{\partial w}$ is the gradient evaluated at the current weight vector and η is the step size which controls how big the step is.
Note on step size:

- Larger η results in bigger steps and thus, care must be taken to not overshoot the minima.
- Smaller η results in smaller steps and too small η can take really long to reach the minima.

Substituting the gradient in equation(16), we get:

$$w = w + \eta \frac{2}{N} X^T(y - Xw) \tag{17}$$

where, $X^T(y - Xw)$ is a vector of residuals (errors) that takes the error on each example, multiplies it with the k^{th} feature and averages it over all the samples.

4.3.1 Cost

Analyzing the cost per iteration in terms of running time we get:

- $Xw = O(kN)$
- $y - Xw = O(N)$
- $X^T(y - Xw) = O(kN)$

Thus, the total cost per iteration is $O(kN)$. This is good (and better than the previous methods) as long as we can converge in a few steps.

4.4 Stochastic Gradient Descent

While batch gradient descent follows the gradient, stochastic gradient descent follows an approximation of the gradient. This can be understood by examining that the gradient as given by

$$\frac{\partial L}{\partial w} = -\frac{2}{N} \sum_{i=1} N(y_i - wx_i)x_i \tag{18}$$

is an average of N observations and therefore, we can just take a subset of the N observations to get a stochastic estimate of the gradient.
Thus, stochastic gradient descent take a random sample set of n observations where $n \leq N$ to estimate the gradient.

$$\frac{\partial L}{\partial w} = -\frac{2}{n} \sum_{i=1} n(y_i - wx_i)x_i \tag{19}$$

Although this is not always exactly equal to the actual gradient, however, it isn't too far off and on an average we usually move in the right direction and eventually reach the minima. In practice, $n = 1$ that is, estimating the gradient using just a single example also works surprisingly well. The step size η, is generally made smaller to guarantee convergence.

4.4.1 Cost

Analyzing the cost per iteration in terms of running time we get:

- $Xw = O(kn)$
- $y - Xw = O(n)$
- $X^T(y - Xw) = O(kn)$

Thus, the total cost per iteration is $O(kn)$. However, we need more iterations now as compared to batch gradient descent.
If we have a lot of samples, stochastic gradient descent works much better, however, with fewer examples it is not a good choice.

4.5 Second Order Derivative methods

In some methods such as the Newton's method, second order derivative of the loss function is calculated to estimate the curvature of the surface. This is because the curvature of the surface can give useful information for tuning the step size. For example, when the surface is flat, we can afford to take big jumps, on the other hand, when the surface is steep we should take smaller steps. However, calculating the second order derivative is computationally expensive and is done only if enough resources are available and if the gradient descent either takes too long or does not converge at all.

5 Additional notes on R Example

- In heavy tail distribution, log-scale is a better option as it can offer more insights into the data and its distribution.
- When the data is heavily skewed, median is more indicative than mean.
- Geometric mean (in log space) and median are very close in terms of numerical value. For this reason, many times the median is estimated by calculating the geometric mean which offers the advantage of being easier to compute and can also be calculated in a streaming setting. It is given by $10^{Mean(\log_{10} x)}$
- Data can be transformed by applying a function $\phi(x)$ without changing anything else.

  $$f(x) = wx \Rightarrow f(x) = w\phi(x) \tag{20}$$

  Depending on $\phi(x)$, $f(x)$ can also be non-linear in x. The squared loss is then given by

  $$L[f] = \frac{1}{N}\sum_{i=1}^{N}(y_i - w\phi(x_i))^2 \tag{21}$$

  which is still quadratic in w and hence, the problem remains the same.

Notes from jjp2172

1 Introduction

Framework for Models:

- Specify outcome and predictors
- Actually a difficult part (usually handed to you)
- Define loss function
- How close model predicts compared with observed data
- Develop algorithm to find the best model
- Mnimize loss function (searching across all possible methods)
- Assess model performance + results

Regressions:

$$Outcomes : \{y_i\}_{i=1}^{N} \tag{22}$$

$$Predictors : \{x_i\}_{i=1}^{N} \quad (Input/Features) \tag{23}$$

X can be a vector of multiple dimensions, or features

Figure 1 k-dimensional vector

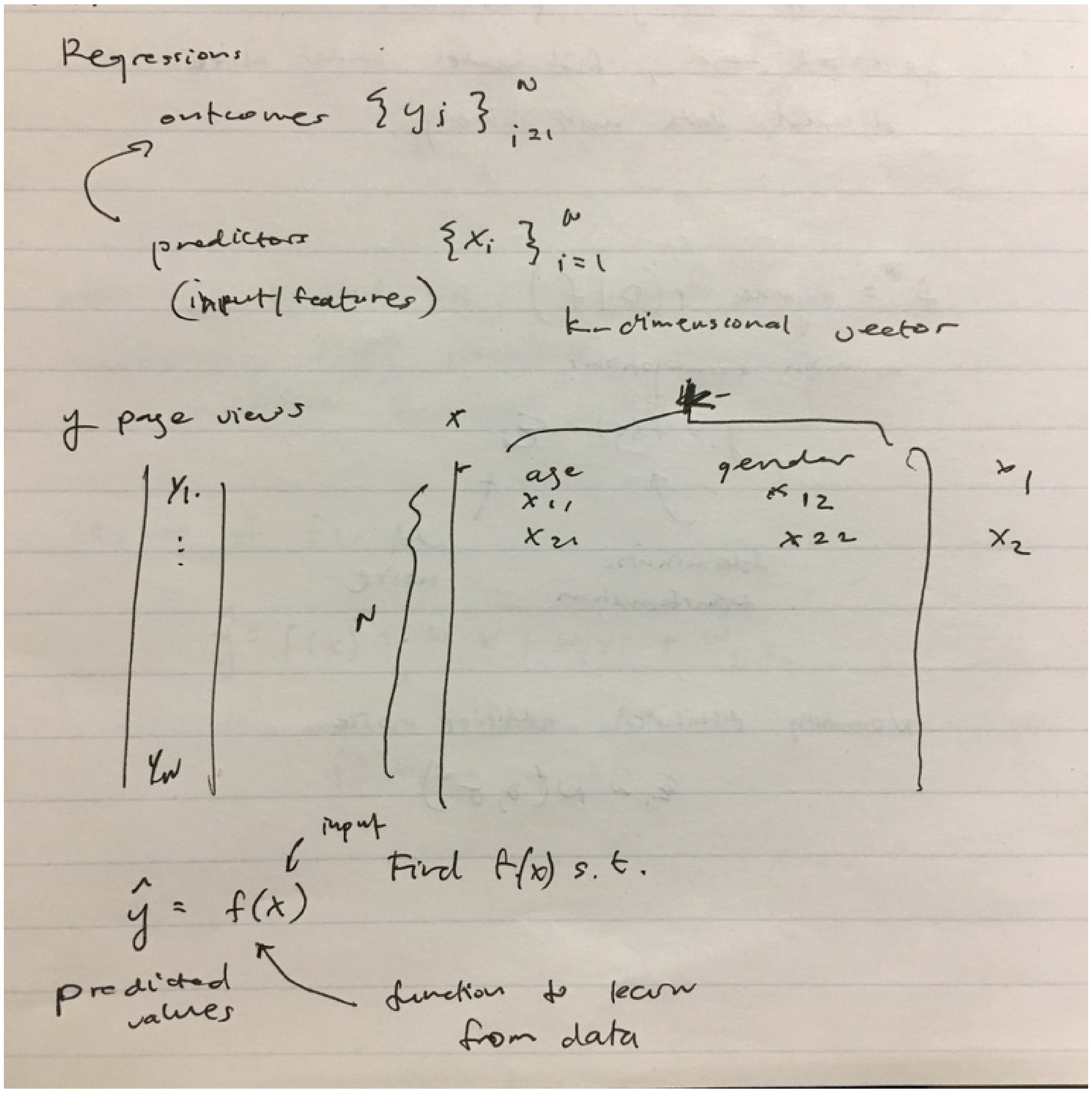
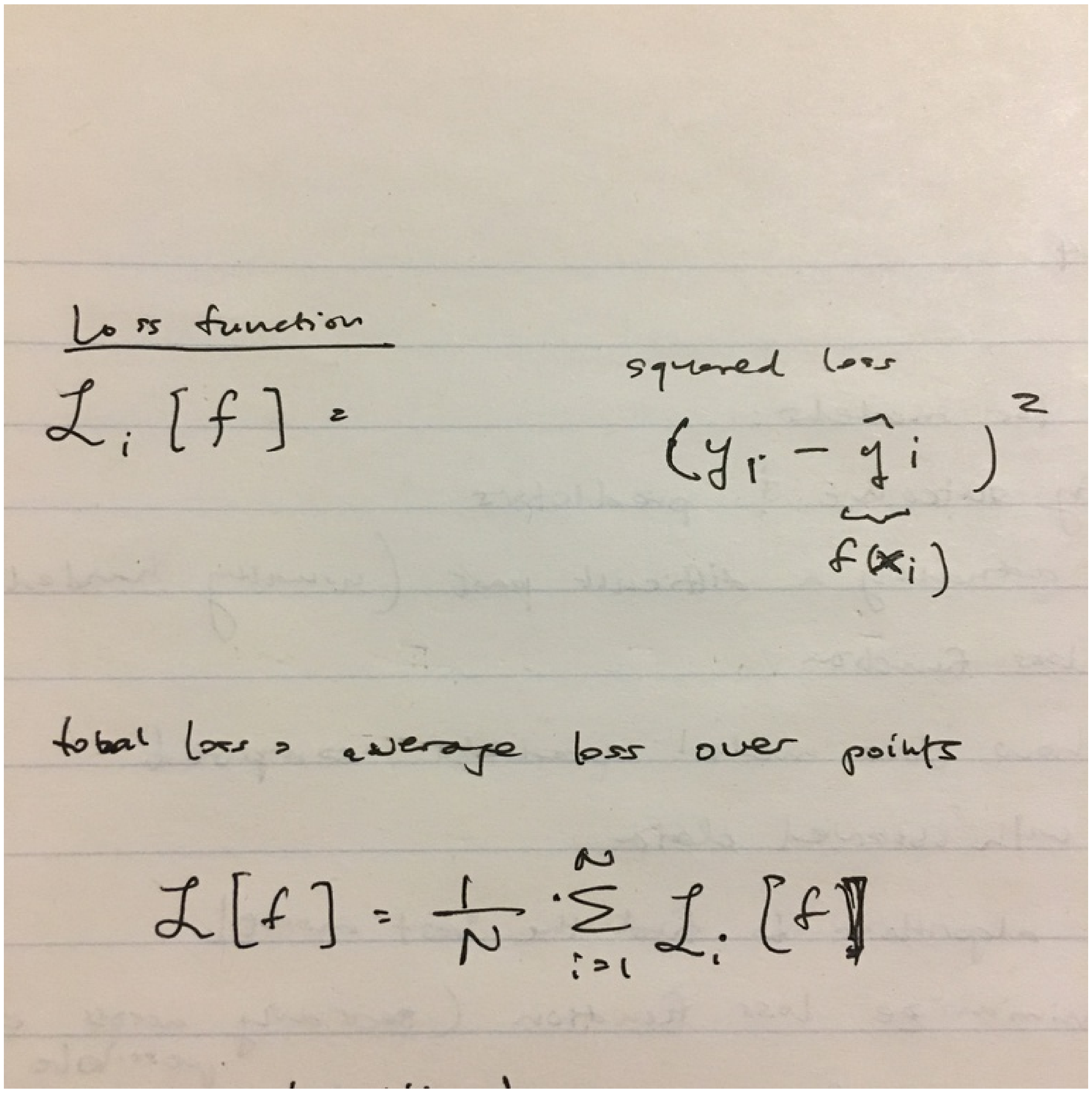


Figure 1: Aim to learn coefficients for vector x to predict y.

Figure 2 Define a loss function to minimize



Loss function

squared loss

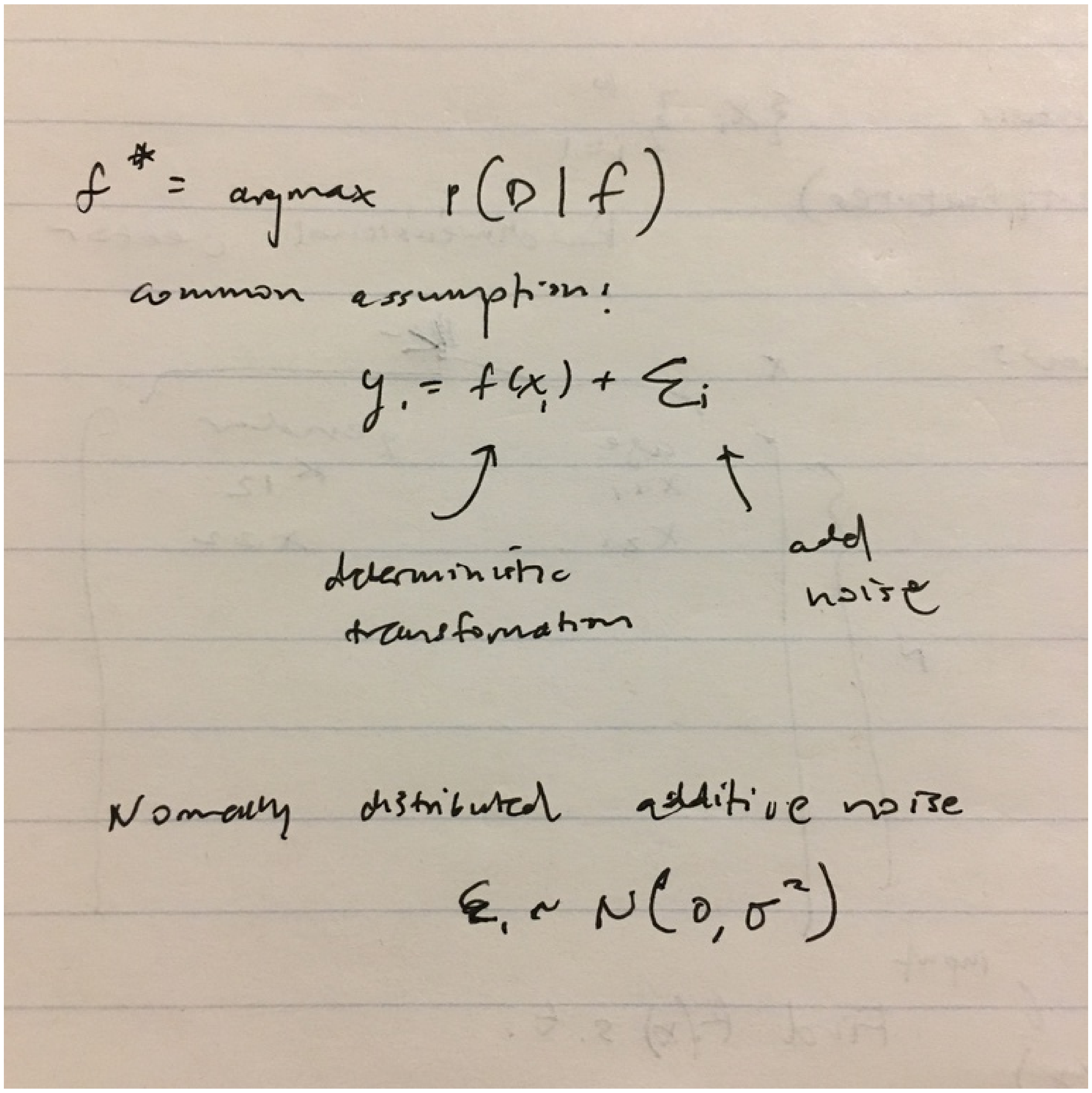
$$\mathcal{L}_i[f] = (y_i - \underbrace{\hat{y}_i}_{f(x_i)})^2$$

total loss = average loss over points

$$\mathcal{L}[f] = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i[f]$$

Figure 2: Minimizing loss function allows to find coefficients with least error.

Figure 3 Maximum Likelihood



$$f^* = \operatorname{argmax}\ p(D \mid f)$$

common assumption:

$$y_i = f(x_i) + \varepsilon_i$$

deterministic transformation

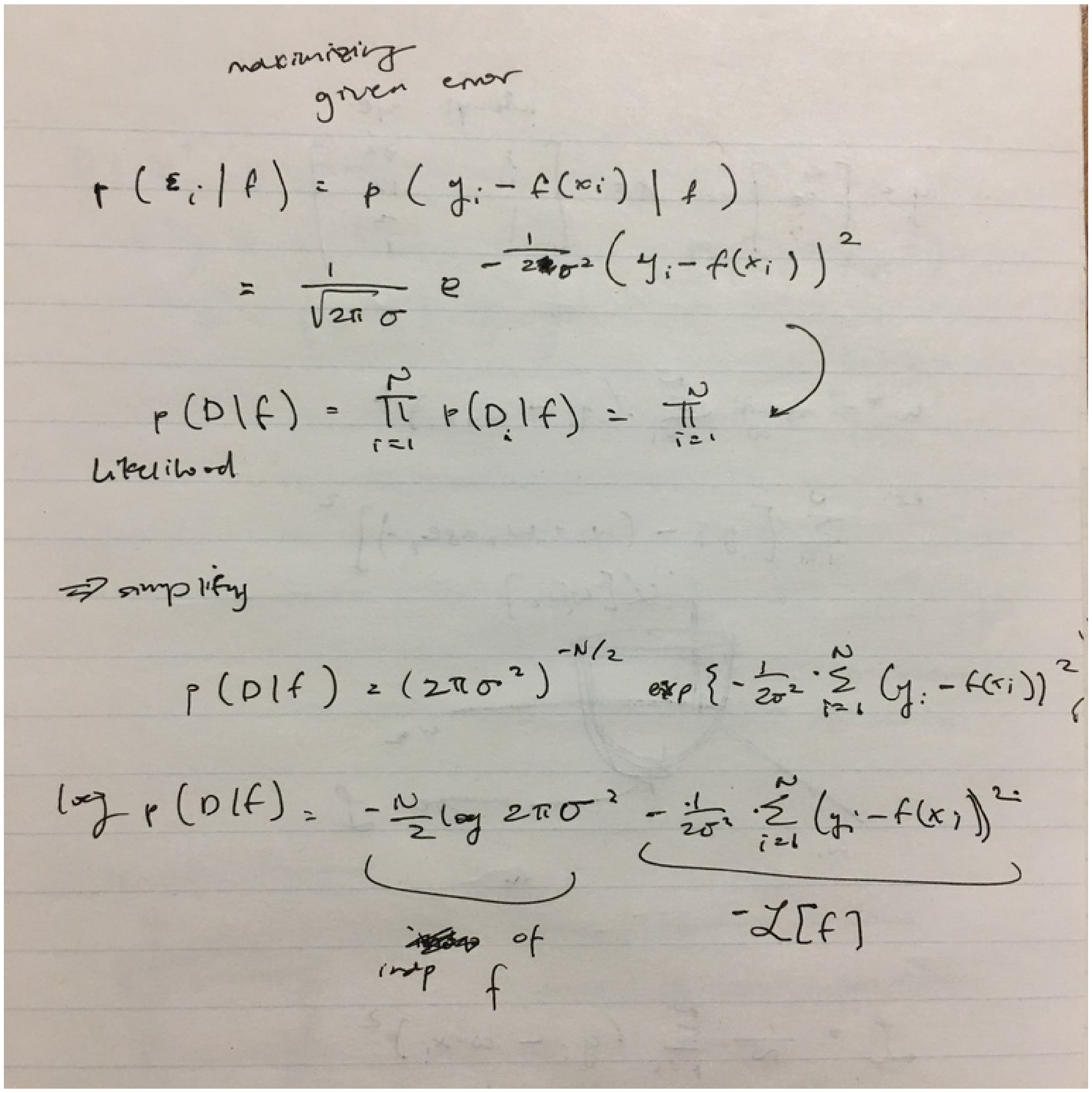
add noise

Normally distributed additive noise

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

Figure 3: Assume some family of probabilistic models generated the data - we find the model under which observed data most likely

Figure 4 Maximizing given error



maximizing given error

$$p(\varepsilon_i | f) = p(y_i - f(x_i) | f)$$

$$= \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{1}{2\sigma^2}(y_i - f(x_i))^2}$$

$$p(D|f) = \prod_{i=1}^{N} p(D_i|f) = \prod_{i=1}^{N}$$

Likelihood

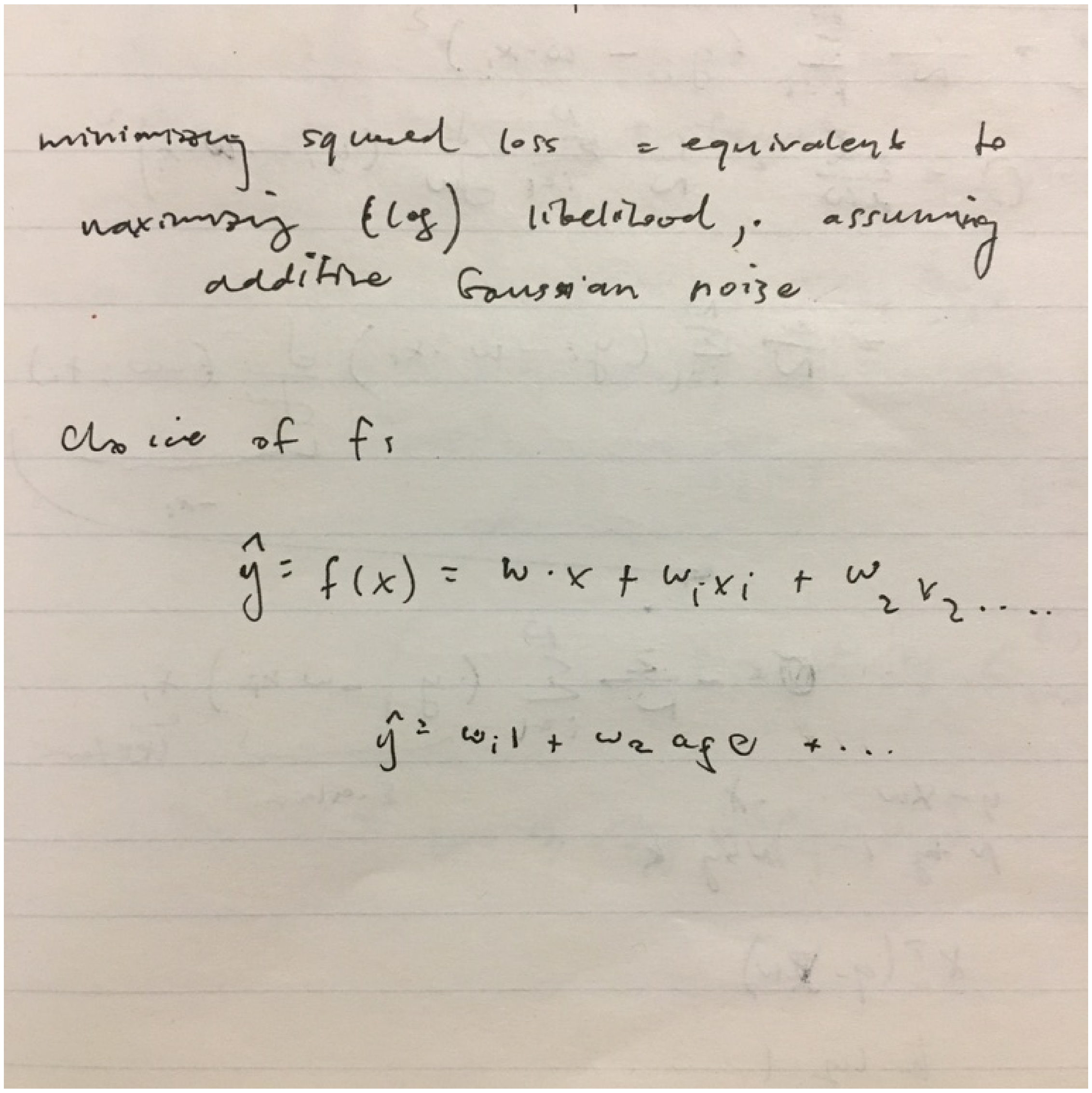
⇒ simplify

$$p(D|f) = (2\pi\sigma^2)^{-N/2} \exp\left\{-\frac{1}{2\sigma^2}\cdot\sum_{i=1}^{N}(y_i - f(x_i))^2\right\}$$

$$\log p(D|f) = \underbrace{-\frac{N}{2}\log 2\pi\sigma^2}_{\text{indep of } f} \underbrace{-\frac{1}{2\sigma^2}\cdot\sum_{i=1}^{N}(y_i - f(x_i))^2}_{-\mathcal{L}[f]}$$

Figure 4: Minimizing squared loss is equivalent to maximizing (log) likelihood, assuming additive Gaussian noise.

Figure 5



minimizing squared loss = equivalent to maximizing (log) likelihood, assuming additive Gaussian noise

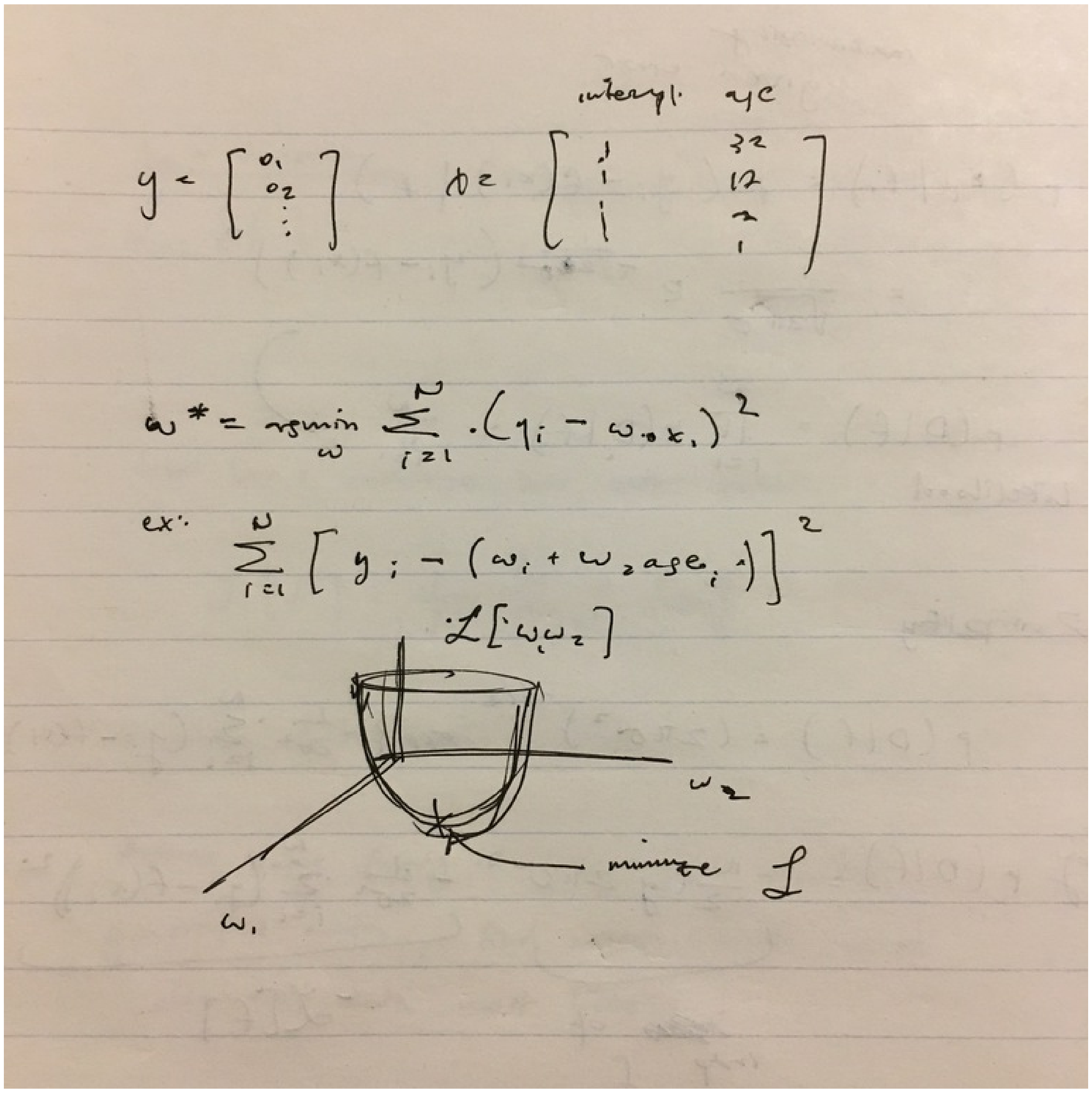
Choice of f:

$$\hat{y} = f(x) = w \cdot x + w_i x_i + w_2 x_2 \ldots$$

$$\hat{y} = w_i 1 + w_2 \, age + \ldots$$

Figure 5: Minimizing squared loss is equivalent to maximizing (log) likelihood, assuming additive Gaussian noise.

Figure 6 Two Dimensional Representation of Loss Function



$$y = \begin{bmatrix} o_1 \\ o_2 \\ \vdots \end{bmatrix} \qquad X = \begin{array}{c} \begin{array}{cc} \text{intercpt.} & \text{age} \end{array} \\ \begin{bmatrix} 1 & 32 \\ 1 & 12 \\ 1 & 2 \\ 1 & 1 \end{bmatrix} \end{array}$$

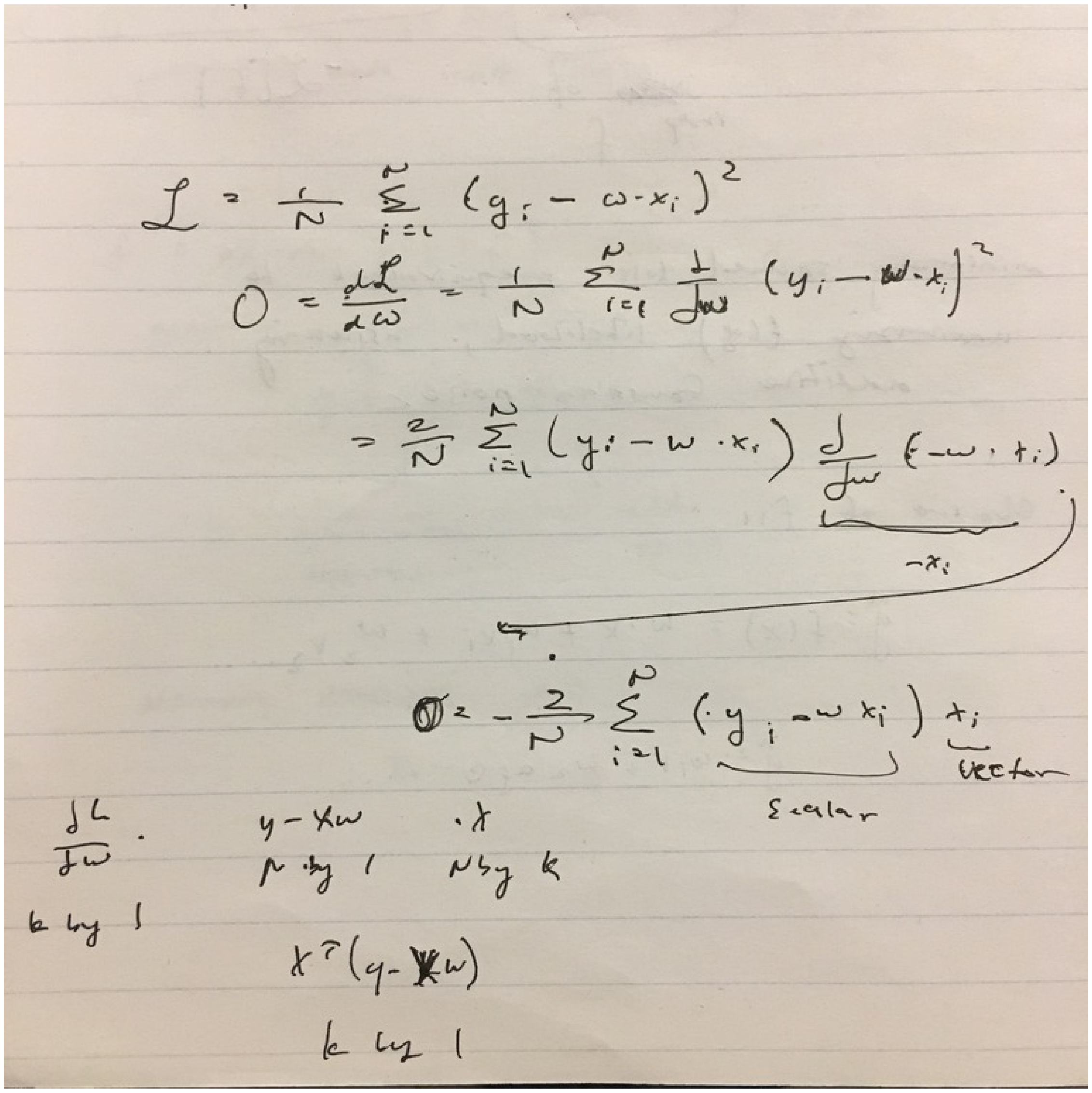
$$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - w \cdot x_i)^2$$

ex: $$\sum_{i=1}^{N} \left[y_i - (w_1 + w_2 \, age_i) \right]^2$$

$$\mathcal{L}[w_1 w_2]$$

Figure 6: If we are working with two dimensions(labels), easier to visualize minimizing L to find coefficients with least loss

Figure 7 Mathematical derivation



$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - w\cdot x_i)^2$$

$$0 = \frac{d\mathcal{L}}{dw} = \frac{1}{N}\sum_{i=1}^{N}\frac{d}{dw}(y_i - w\cdot x_i)^2$$

$$= \frac{2}{N}\sum_{i=1}^{N}(y_i - w\cdot x_i)\underbrace{\frac{d}{dw}(-w\cdot x_i)}_{-x_i}$$

$$0 = -\frac{2}{N}\sum_{i=1}^{N}\underbrace{(y_i - w\, x_i)}_{\text{Scalar}}\underbrace{x_i}_{\text{vector}}$$

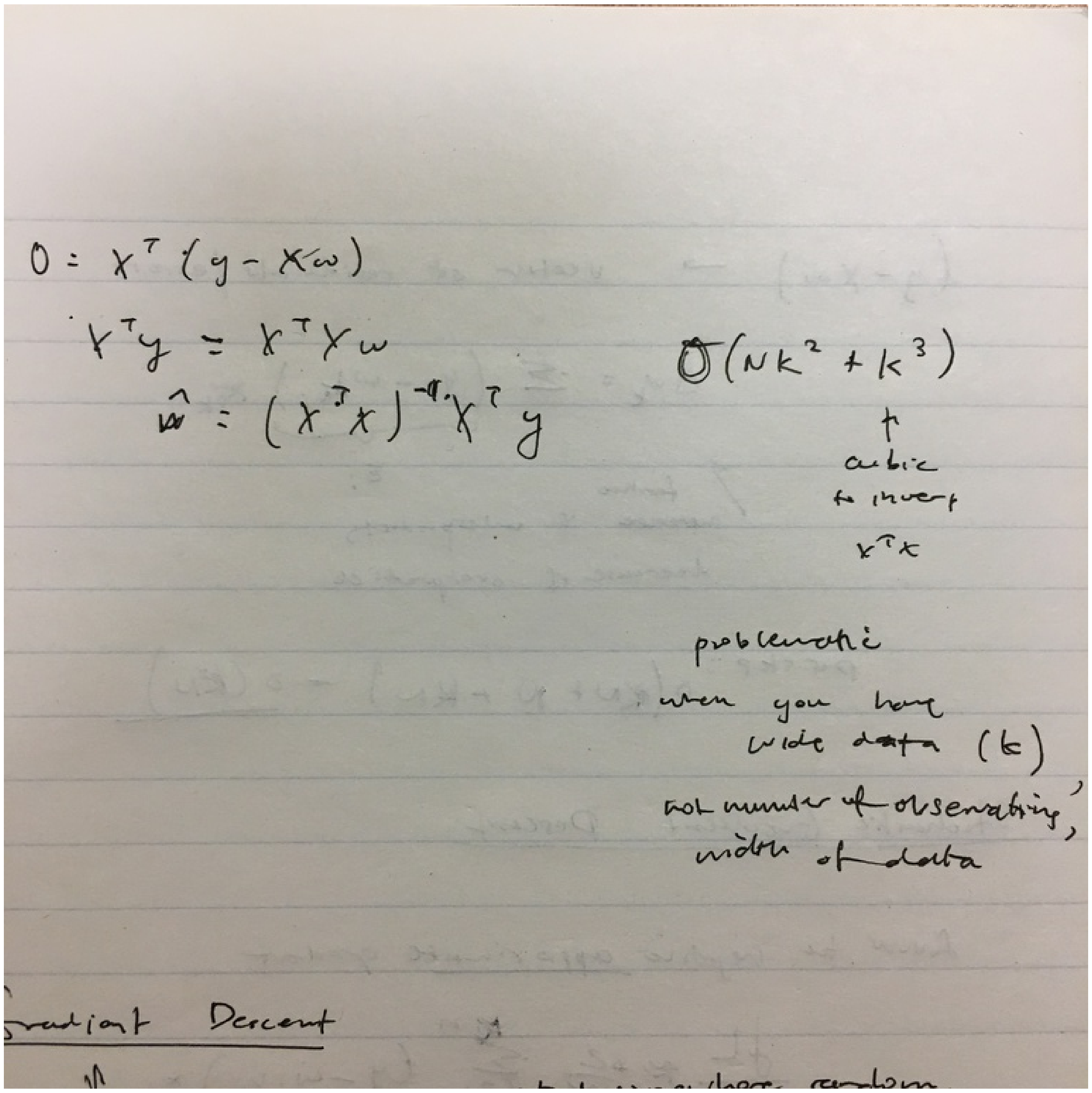
$\frac{dL}{dw}$	$y - Xw$	$\cdot X$
	N by 1	N by k
k by 1		

$$X^T(y - \cancel{X}w)$$

k by 1

Figure 7: Take derivative of loss function (averaged over all values) and set to 0 to solve for our vector of coefficients.

Figure 8 Converting to matrix operations



$$0 = X^T (y - Xw)$$

$$X^T y = X^T X w$$

$$\hat{w} = (X^T X)^{-1} X^T y$$

$$O(Nk^2 + k^3)$$

$\uparrow$ cubic to invert $X^T X$

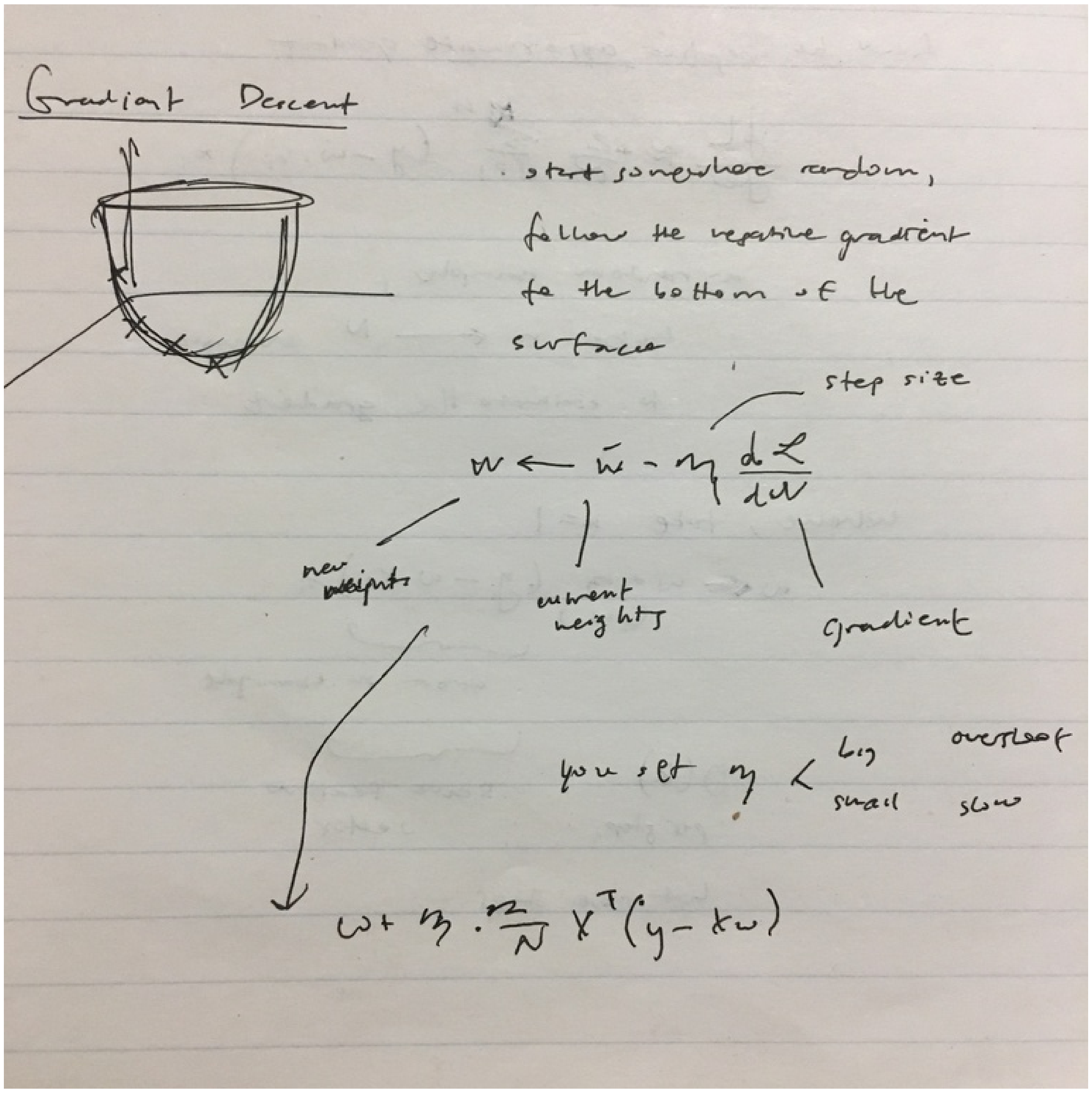
problematic when you have wide data (k)

not number of observations, width of data

Gradient Descent

Figure 8: Since we are working with vectors, we can use linear algebra and matrix multiplication to solve for our coefficient vector w. Note, it becomes problematic when we have wide data (high dimensions k) as matrix operations are extremely expensive in this manner - number of observations is less impactful on runtime than width of data

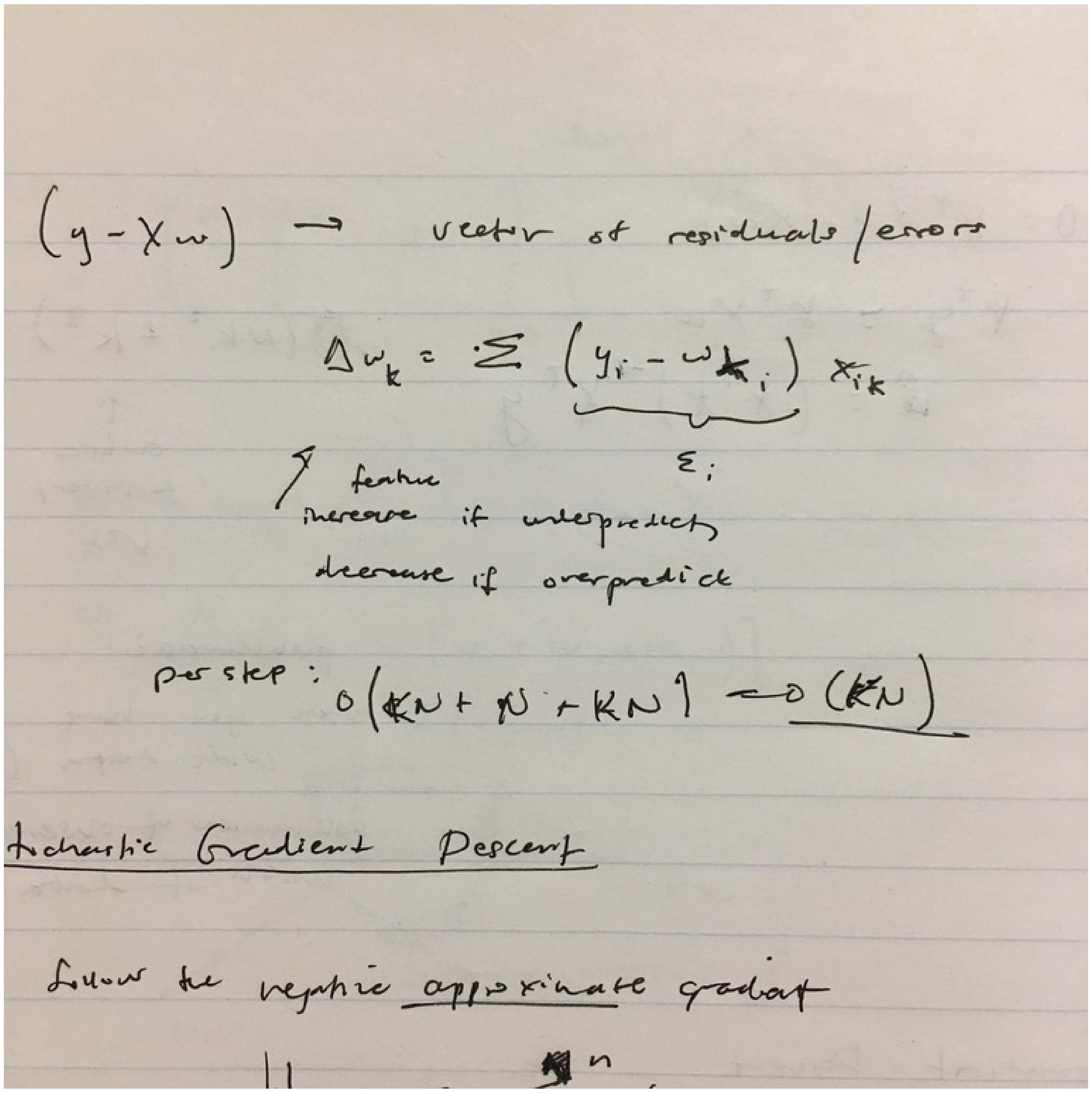
Figure 9 Gradient Descent



Gradiant Descent
· start somewhere random, follow the negative gradient to the bottom of the surface
step size
$w \leftarrow \bar{w} - \eta \frac{d\mathcal{L}}{dw}$
new weights
current weights
gradient
you set η < big overshoot / small slow
$w + \eta \cdot \frac{2}{N} X^T(y - Xw)$

Figure 9: Start somewhere random, follow the negative gradient to the bottom of the surface. You set the learning rate (can be variable) - too big and it will overshoot the minimum, too small and gradient descent will take a long time

Figure 10 Vector of residuals/errors



$(y - Xw) \rightarrow$ vector of residuals/errors

$$\Delta w_k = \sum \underbrace{(y_i - w x_i)}_{\varepsilon_i} x_{ik}$$

↗ feature
increase if underpredict
decrease if overpredict

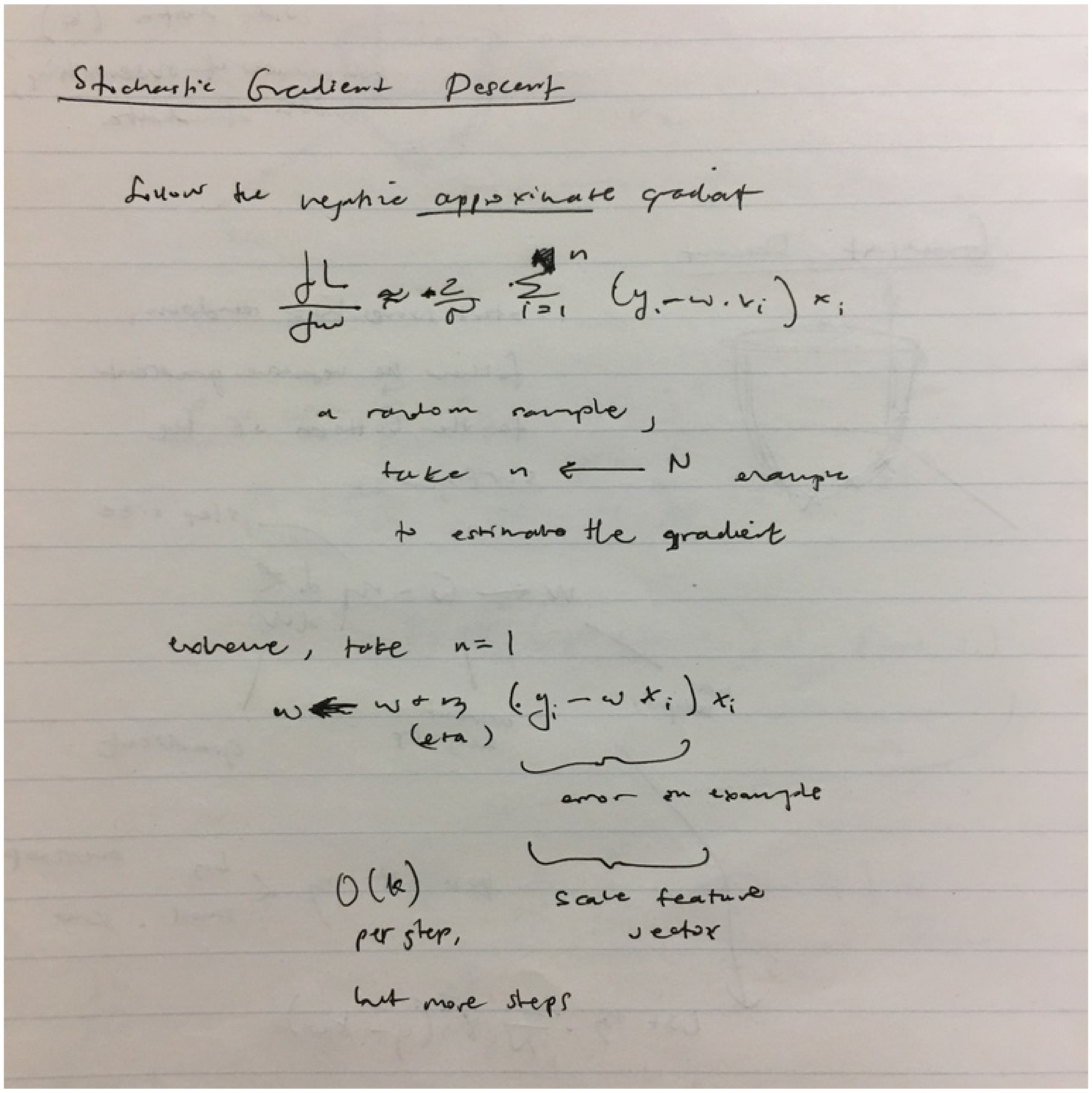
per step: $O(KN + N + KN) \rightarrow \underline{O(KN)}$

Stochastic Gradient Descent

Follow the negative approximate gradient

Figure 10: Increase features w if underpredict, decrease features w if overpredict

Figure 11 Stochastic Gradient Descent



Stochastic Gradient Descent

Follow the negative approximate gradient

$$\frac{dL}{dw} \approx \frac{2}{n} \sum_{i=1}^{n} (y_i - w \cdot x_i)\, x_i$$

a random sample,

take $n \leftarrow N$ example

to estimate the gradient

extreme, take $n=1$

$$w \leftarrow w + \eta \underbrace{(y_i - w\, x_i)}_{\text{error on example}} \underbrace{x_i}_{\text{scale feature vector}}$$

(eta)

$O(k)$ per step,

but more steps

Figure 11: Follow the negative approximate gradient - take an n random sample to estimate the gradient

Notes from td2520

1 Introduction

1.1 What is Regression?

- best-fit line
- best-fit curve
- finding patterns in the data
- function we learn

1.2 Formal Definition:

"...to understand as far as possible with the available data how the conditional distribution of the response y varies across subpopulations determined by the possible values of the predictor or predictors."

1.3 Purpose of Regression

- Describe - compact summary of outcomes
- Predict - future of unobserved conditions
- Explain - associations, interpret effects of coefficients.

1.4 Goal:

Flexible models to describe what we have seen before simple enough to generalize future outcome.

1.5 Framework:

- Specify the outcome + predictors, and form of the model relating them
- Define a loss fn. that quantifies how close a model's prediction are to observed outcomes.
- Develop an algorithm to fit the model to the observations by minimizing this loss.
- Assess model performance + interpret results.

2 Math

Y is a vector of outcome.
X is a matrix of input, each column corresponds to a feature (age,gender) as shown in Fig. 1
Loss function : how well function f predicts the outputs from inputs
$L_i[f] = (y - \hat{y})^2$

$$L[f] = 1/N * \sum_{i=1}^{n} L_i[f] \tag{24}$$

$$L[f] = 1/N * \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{25}$$

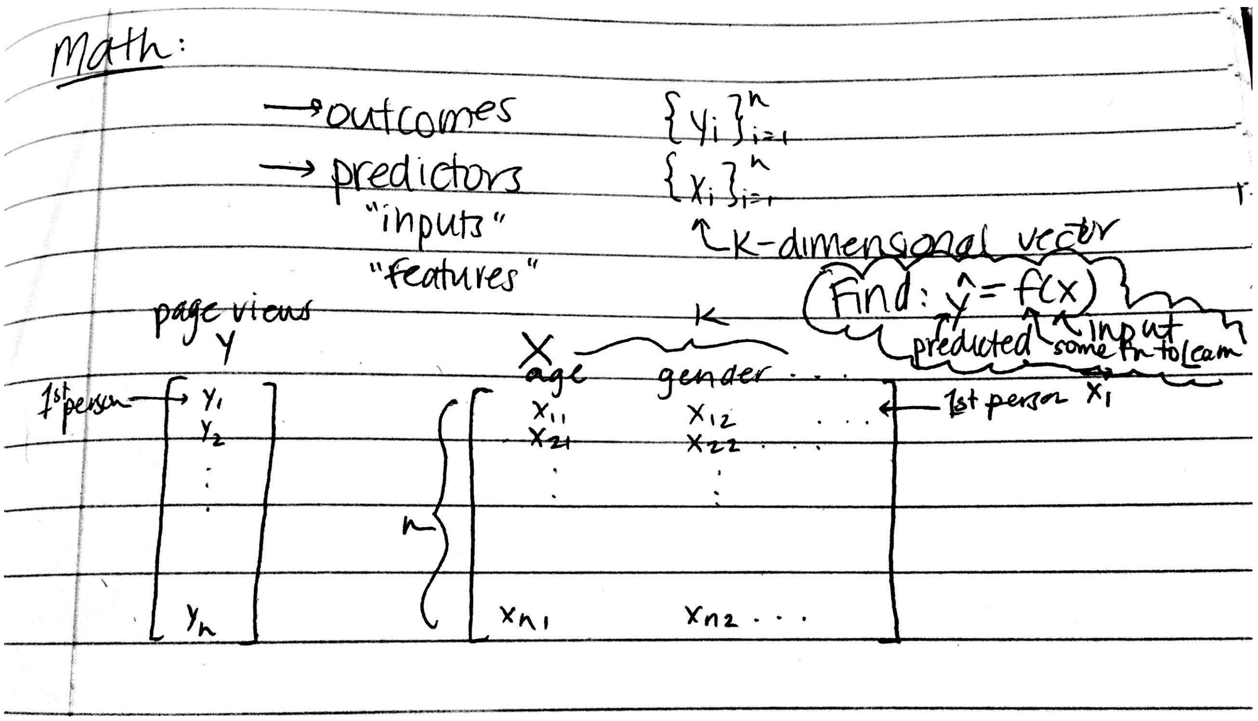


Figure 12: Representing the data for Section Math

2.1 Maximum Likelihood

:

Assume some family of probabilistic model generated the data. Find the model under which the observed data are most likely.

f* = argmax$_f$ P($D|f$)

Common assumption : y =f(x$_i$) + E$_i$

P($E_i|f$) = P($y_i - f(x_i)|f$)

= (1 / $\sqrt{\ } 2\pi\sigma^2$) * exp$^{(-1/2\sigma^2)*(y_i - f(x_i))^2}$

$$P(D|f) = \prod_{i=1}^{n} P(D_i|f) \tag{26}$$

$$P(D|f) = \prod_{i=1}^{n} (1/\sqrt{2}\pi\sigma^2) * exp^{(-1/2\sigma^2)*(y_i - f(x_i))^2} \tag{27}$$

Simplifying Likelihood Fn.

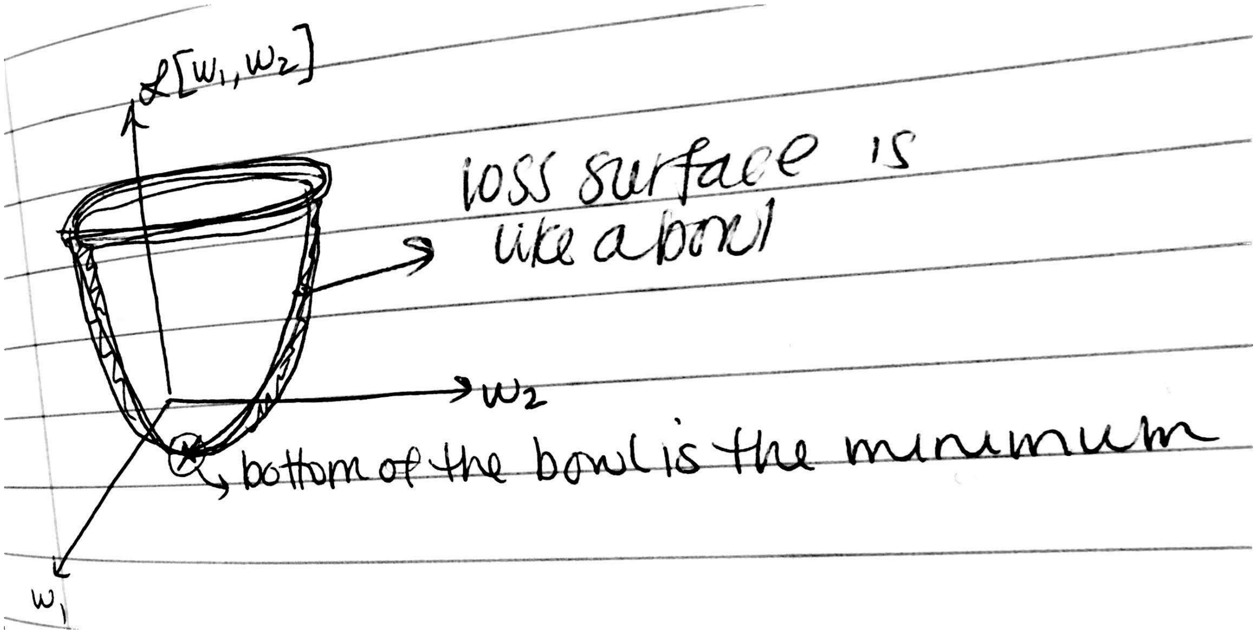
$$P(D|f) = (2\pi\sigma^2)^{-n/2} * exp(-1/2\sigma^2 * \sum_{i=1}^{n}(y_i - f(x_i))^2) \tag{28}$$

$$LogP(D|f) = -n/2 * (2\pi\sigma^2) - 1/2\sigma^2 * \sum_{i=1}^{n}(y_i - f(x_i))^2 \tag{29}$$

Maximizing squared loss = maximising (log) likelihood, assuming additive gaussian noise. (errors are normally distributed)

Choice for f: $\hat{y} = f(x) = w.X = w_1x_1 + w_2x_2 + ... + w_nx_n$

$$w^* = argmin_w \sum_{i=1}^{n} (y_i - w.x_i) \tag{30}$$



$\mathcal{L}[w_1, w_2]$
loss surface is like a bowl
w_2
bottom of the bowl is the minimum
w_1

Figure 13: Loss Surface

Loss Surface is easy to visualise in 2D, difficult with high dimensions. One shouldnt do a manual search for the bottom of the loss surface

$$L[f] = 1/N * \sum_{i=1}^{n} (y_i - w.x_i)^2 \tag{31}$$

Differentiating -(8) wrt w and equating with 0.

$$0 = -2/N * \sum_{i=1}^{n} (y_i - w.x_i)x_i \tag{32}$$

Equation -(9) can be written in vectorized notation as :

$$0 = X^T * (y - Xw) \tag{33}$$

Simplifying:

$\hat{w} = (X^TX)^{-1}X^Ty$

$\hat{w}$ is the best estimate of w. Time complexity to find $\hat{w}$ is $O(NK^2 + K^3)$

If the data is very wide, this would be very time consuming.

2.2 Gradient Descent

WRT to Loss Surface

1. Start somewhere random
2. Follow the negative gradient to the bottom of the surface.

   w = w - eta * (gradient)
   gradient is same as equation - (9)

   eta is the step size: large = big steps, small = slow progression

Computational Costs:
Xw = O(KN)
Y-Xw=O(N)
X^T(Y-Xw) =O(KN)
Time Complexity is O(KN + N + KN) = O(KN)

Gradient Descent would be better for higher dimension.

2.3 Stochastic Gradient Descent

Same intuition (follow the gradient) but follow the approximate gradient.

$$gradientofLossfunction(dL/dw) = -2/N * \sum_{i=1}^{n}(y_i - w.x_i)x_i \tag{34}$$

Take a random sample n very very small as compared to N examples to estimate the gradient.

$$gradientofLossfunction(dL/dw) = -2/n * \sum_{i=1}^{n}(y_i - w.x_i)x_i \tag{35}$$

Time Complexity is O(KN + N + KN) = O(KN)

On avg you are correct + close to the bottom. Choosing eta becomes tricky, make it smaller as you go closer to convergence

2.4 2nd order derivative

In methods like newton method, second order derivative of loss function is calculated to determine the curvature of the surface. More computation to take 2nd order derivative but saves time to reach to minimum. Faster steps on flatter surface and smaller steps on steep surface.

2.5 Some Additional Points

1. In highly skewed data, used median over mean.
2. Median and Geometric Mean in log scale are pretty close.
3. Log scale is preferred while working with heavy tail distribution.

Notes from yl3156

1 Introduction

1.1 Definition

Regression analysis is the process of estimating the relationships among variables. The goal of using regression analysis is to provide a compact summary of available data, forcast future data, and explain the relationships between variables. Ultimately, we would want to strike a balance between being able to describe observed data and predicting future data.

1.2 Framework

We need the following steps to make a good regression analysis on a data set.

1. Make sound decisions about the outcome, predictor, and form of the model.
2. Define a loss function to describe how close the model's prediction is with respect to the observed data.
3. Devise an algorithm to minimize loss.
4. Define metrics to assess model performance and be able to interpret results.

2 Regression Models

2.1 Task

We have inputs k-dimensional vector x_i and scalar y_i for data points $i = 1, \dots, N$, and would like to find model f such that $\hat{y} = f(x)$ by learning parameters of the model from data.

2.2 Loss Function

We have many different kinds of loss functions. These functions usually outputs some kind of difference between the predicted outcome and the actual outcome. Common loss functions include:

Absolute loss function $L[f] = \frac{1}{N}\sum_1^N |y_i - f(x_i)|$

Squre loss function $L[f] = \frac{1}{N}\sum_1^N (y_i - f(x_i))^2$

2.3 Maximum Likelyhood Interpretation

Assume some family of probablistic model generated the data, then find the model under which the observed data are most likely to occur.
If we assume that $y_i = f(x_i) + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2)$, then we can have the following:

$$p(\epsilon_i|f) = p(y_i - f(x_i)|f) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y_i - f(x_i))^2}$$

Thus we can write the likelyhood for the entire data set as the following:

$$p(D|f) = \prod_i^N p(D_i|f) = \prod_i^N \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y_i - f(x_i))^2}$$

Taking log probability this becomes:

$$\log p(D|f) = -k_1 N - k_2 \sum_1^N (y_i - f(x_i))^2$$

where k_1 and k_2 are some constants. Since $-k_2 \sum_1^N (y_i - f(x_i))^2 = -k_2 L[f]$ where L is the square loss function, we have shown that minimize the square loss of a data set is equivalent to maximizing the log likelyhood of that data set.

2.4 Model Parameters

Suppose $\hat{y} = f(x) = w \cdot x$ where w is the set of parameters for the model and x is the set of features of the data. We would like to devise a way to find w that minize the loss over the entire data set. Let $L = \frac{1}{N} \sum_1^N (y_i - w \cdot x_i)^2$, we take the derivative of L and set it to 0:

$$\frac{\partial L}{\partial w} = \frac{1}{N} \sum_1^N \frac{\partial}{\partial w} (y_i - w \cdot x_i)^2 = \frac{2}{N} \sum_1^N (y_i - w \cdot x_i) \frac{\partial}{\partial w} (w \cdot x_i) = \frac{2}{N} \sum_1^N (y_i - w \cdot x_i) x_i = 0$$

Then we can write this equation in matrix form as:

$$\frac{\partial L}{\partial w} = X^T (y - Xw) = 0$$

and we finally derive the normal equation:

$$\hat{w} = (X^T X)^{-1} X^T y$$

Computing this closed form solution, however, requires $O(Nk + k^3)$ runtime. This is very computationally expensive if k, the number of dimensions for the input feature, is large, making this method impractical for high dimensional data.

3 Gradient Descent Methods

Luckily we have iterative methods to find the best model parameters. The idea is to start at a random place in the parameter space, then follow the negative gradient to the bottom of the bowl shaped surface. We introduce three versions of this method.

3.1 Gradient Descent

We use

$$w := w - \alpha \frac{\partial L}{\partial w}$$

which is

$$w := w - \alpha \frac{2}{N} X^T (y - Xw)$$

as the update rule at each step. Parameter α is called the learning rate, which is the step size we take at each iteration in the direction of negative gradient. This method take $O(kN)$ to compute the update for w at each step, which is still very expensive if k is large.

3.2 Batch Stochastic Gradient Descent

We could use a randomly selected subset of the data to update the model parameters at each step instead of the whole set of data. So the update rule becomes:

$$w := w - \alpha \frac{2}{n} \sum_1^n (y_i - w \cdot x_i) x_i$$

where n is a much smaller number than N. This computes the approximate gradient at a cost of $O(kn)$ per step. If n is small enough, this method could make a hugh difference in terms of runtime.

3.3 Stochastic Gradient Descent

We could also use a single randomly selected data point to update parameters in each iteration, using the following update rule:

$$w := w - \alpha(y_i - w \cdot x_i)x_i$$

This method takes $O(k)$ and often can give the model parameters close enough to the optimal ones in much less time than the first gradient decnet method in practice.