Lecture 6: Regression I: Theory and Practice
Modeling Social Data, Spring 2017
Columbia University

Jin Peng

March 2, 2017

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:

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

$$Predictors : \{x_i\}_{i=1}^{N} \qquad (Input/Features) \tag{2}$$

X can be a vector of multiple dimensions, or features

Figure 1 k-dimensional vector

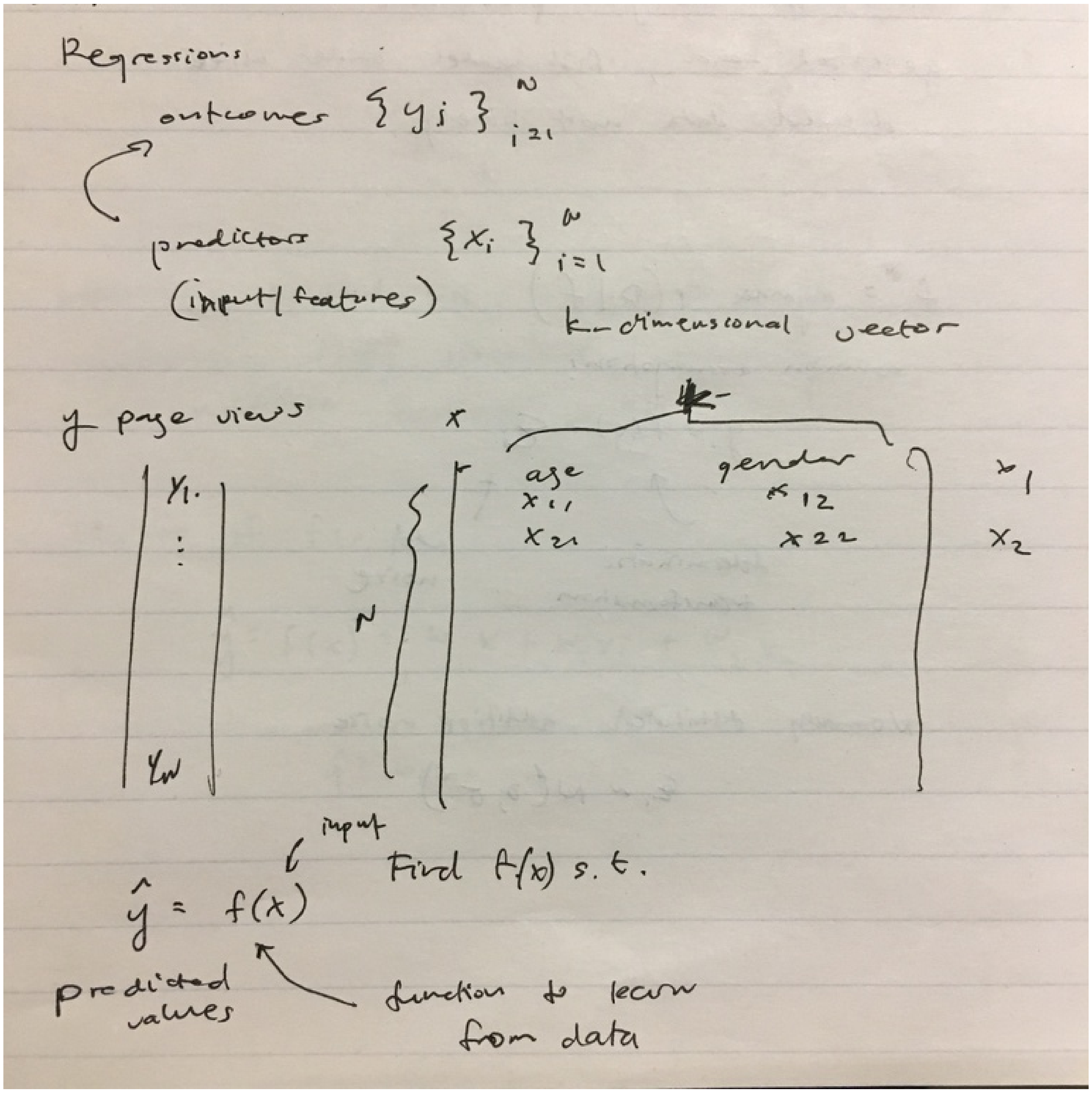

Figure 1: Aim to learn coefficients for vector x to predict y.

Figure 2 Define a loss function to minimize

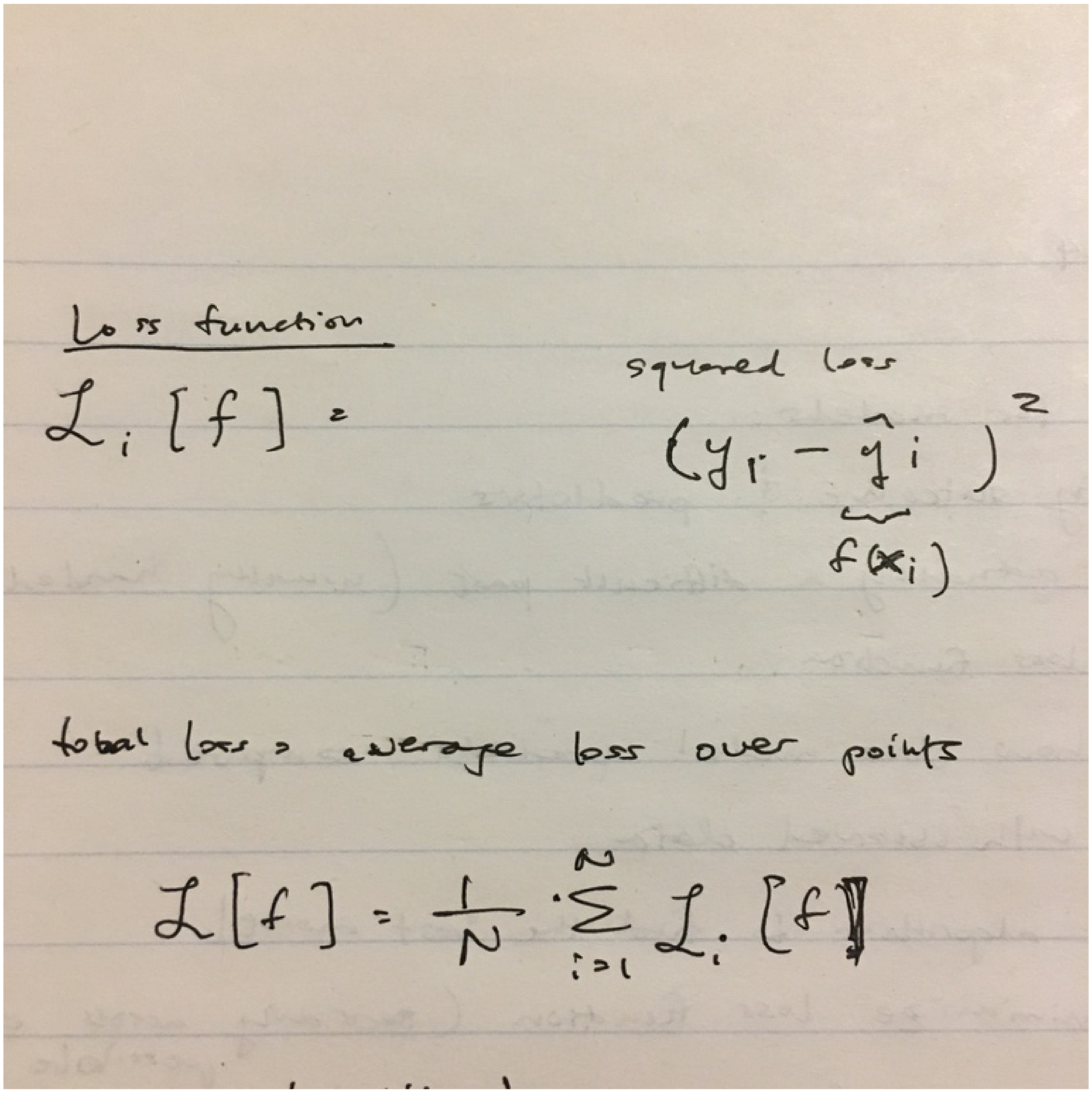

Figure 2: Minimizing loss function allows to find coefficients with least error.

Figure 3 Maximum Likelihood

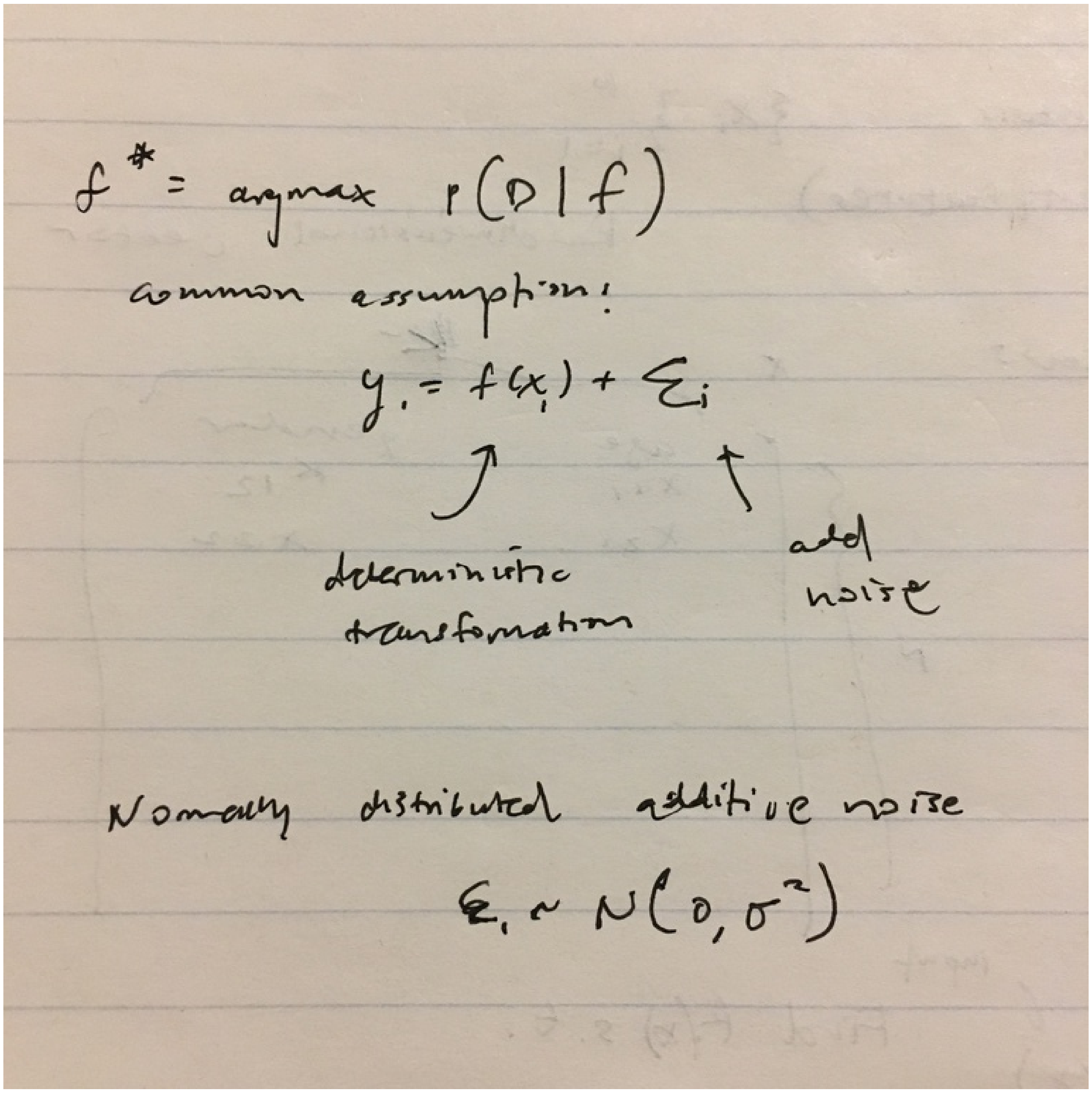

Figure 3: Assume some family of probabilistic models generated the data - we find the model under which observed data most likely

Figure 4 Maximizing given error

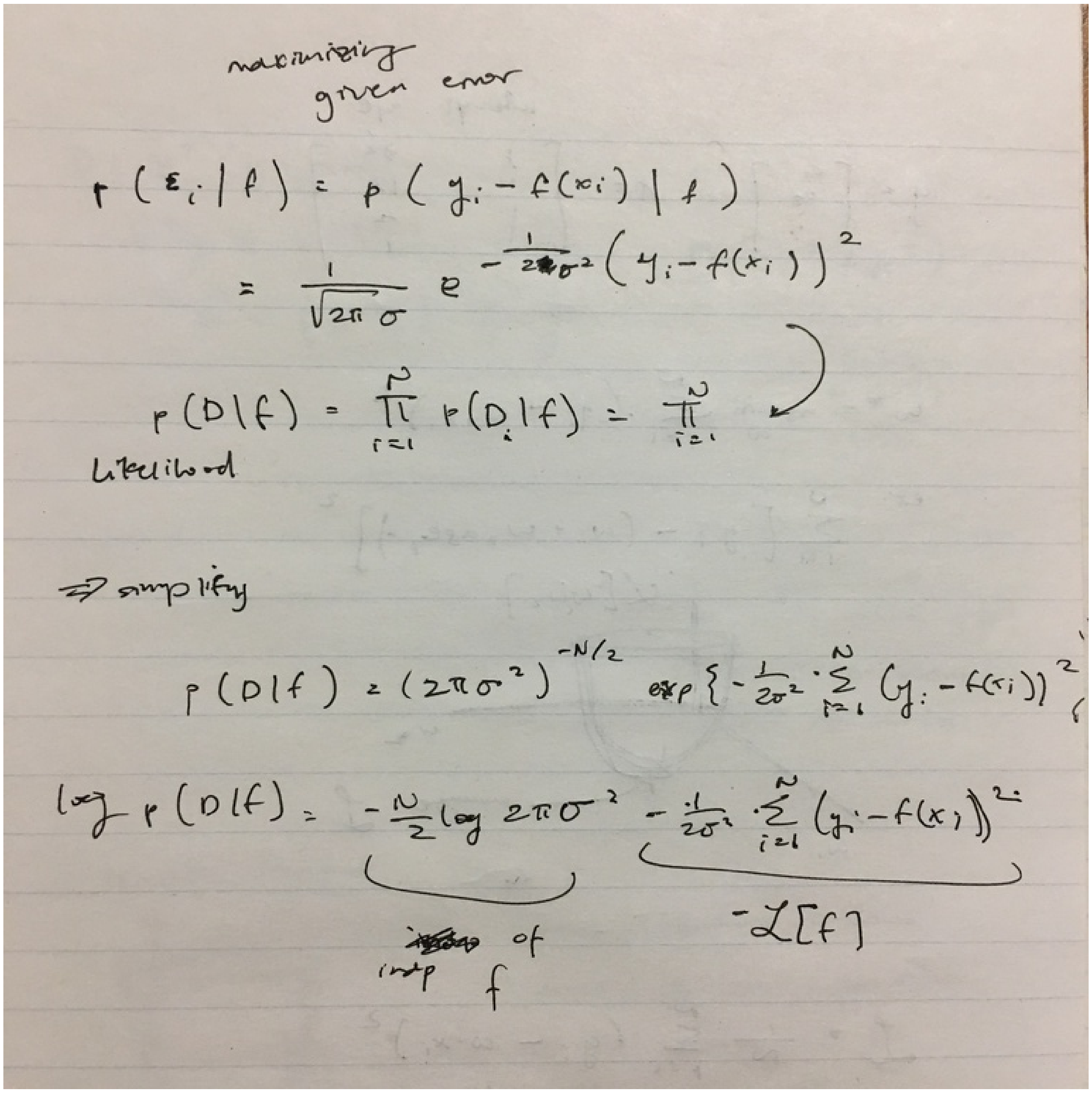

Figure 4: Minimizing squared loss is equivalent to maximizing (log) likelihood, assuming additive Gaussian noise.

Figure 5

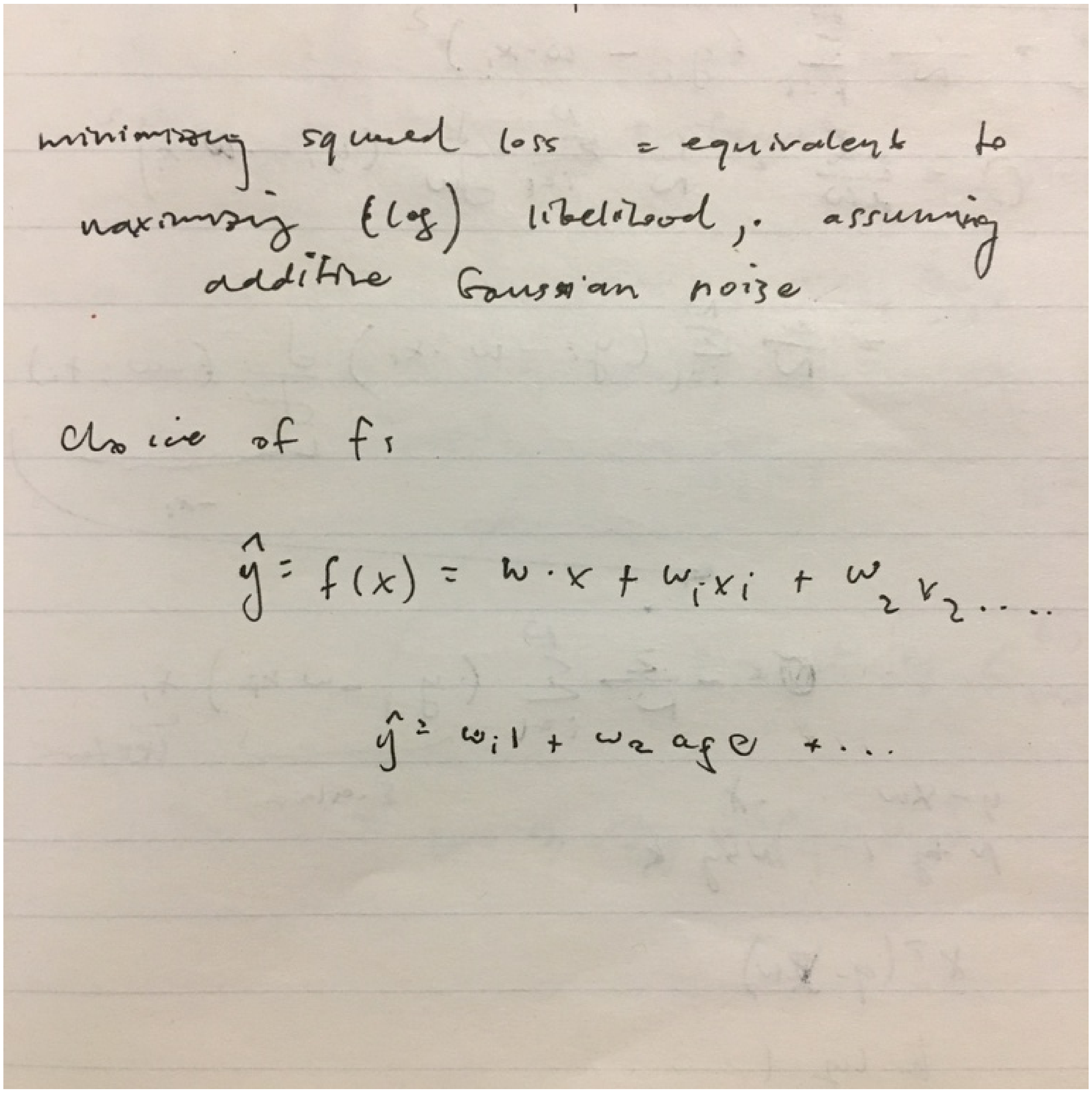

Figure 5: Minimizing squared loss is equivalent to maximizing (log) likelihood, assuming additive Gaussian noise.

Figure 6 Two Dimensional Representation of Loss Function

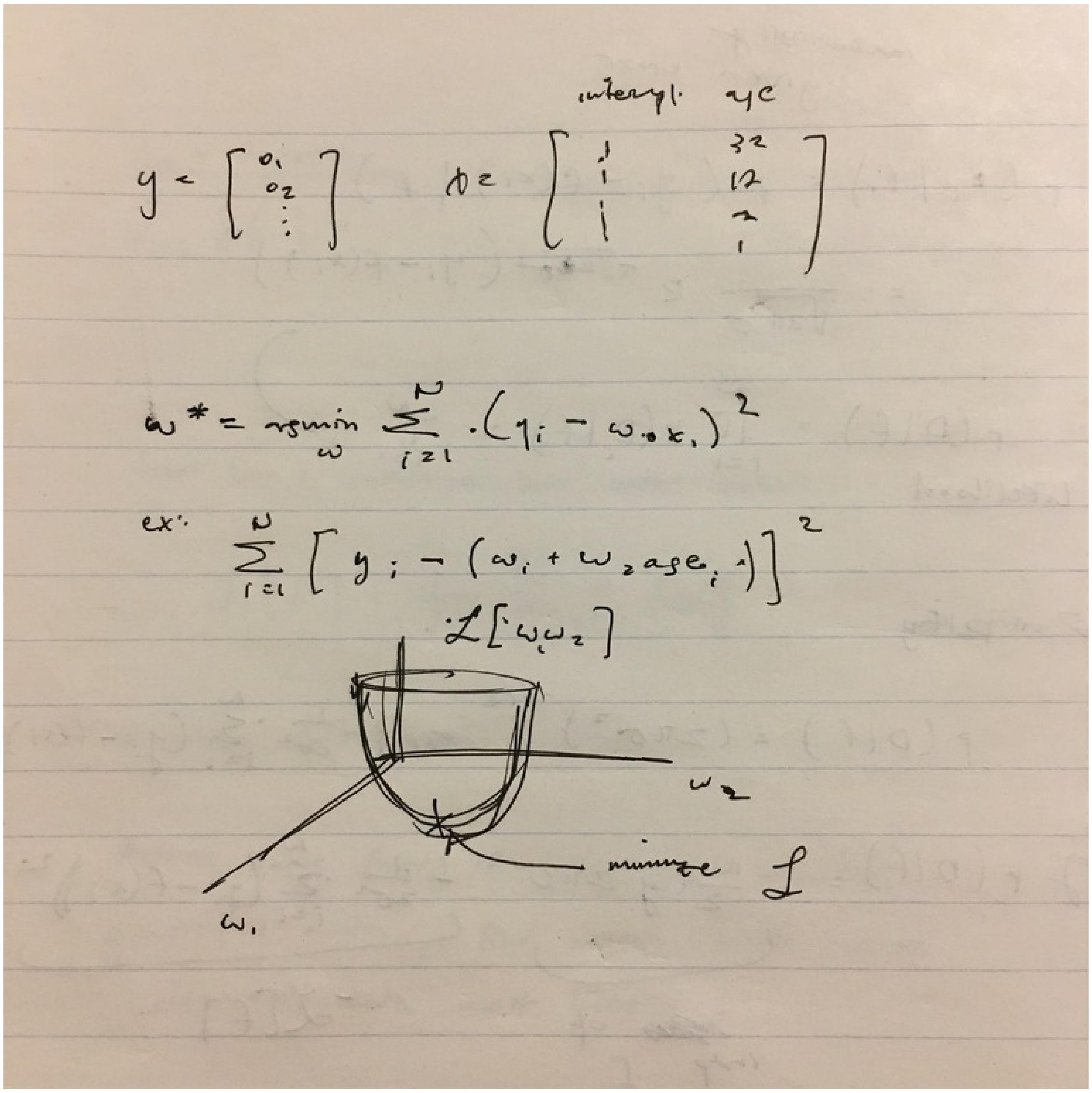

Figure 6: If we are working with two dimensions(labels), easier to visualize minimizing L to find coefficients with least loss

Figure 7 Mathematical derivation

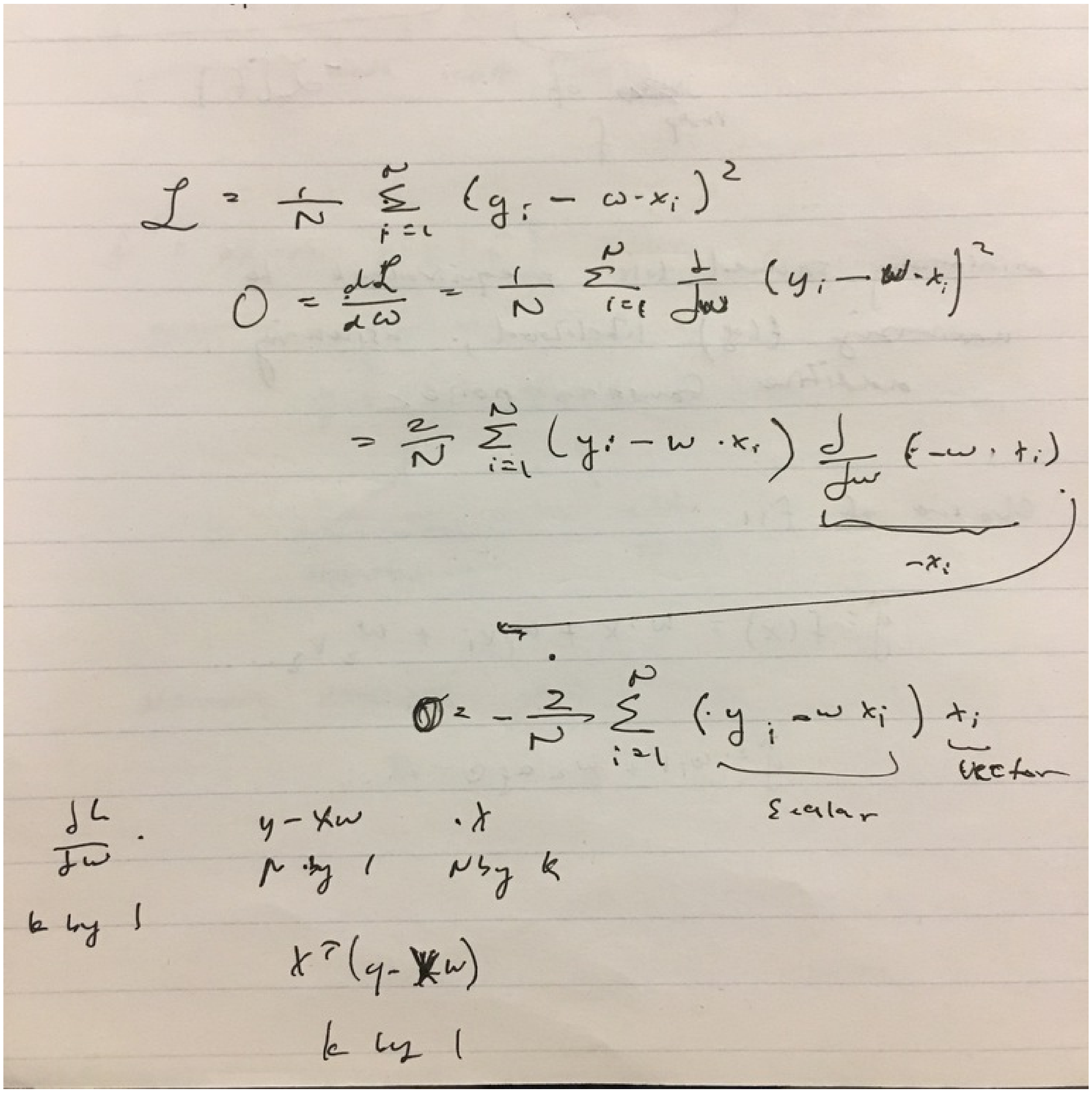

Figure 7: Take derivative of loss function (averaged over all values) and set to 0 to solve for our vector of coefficients.

Figure 8 Converting to matrix operations

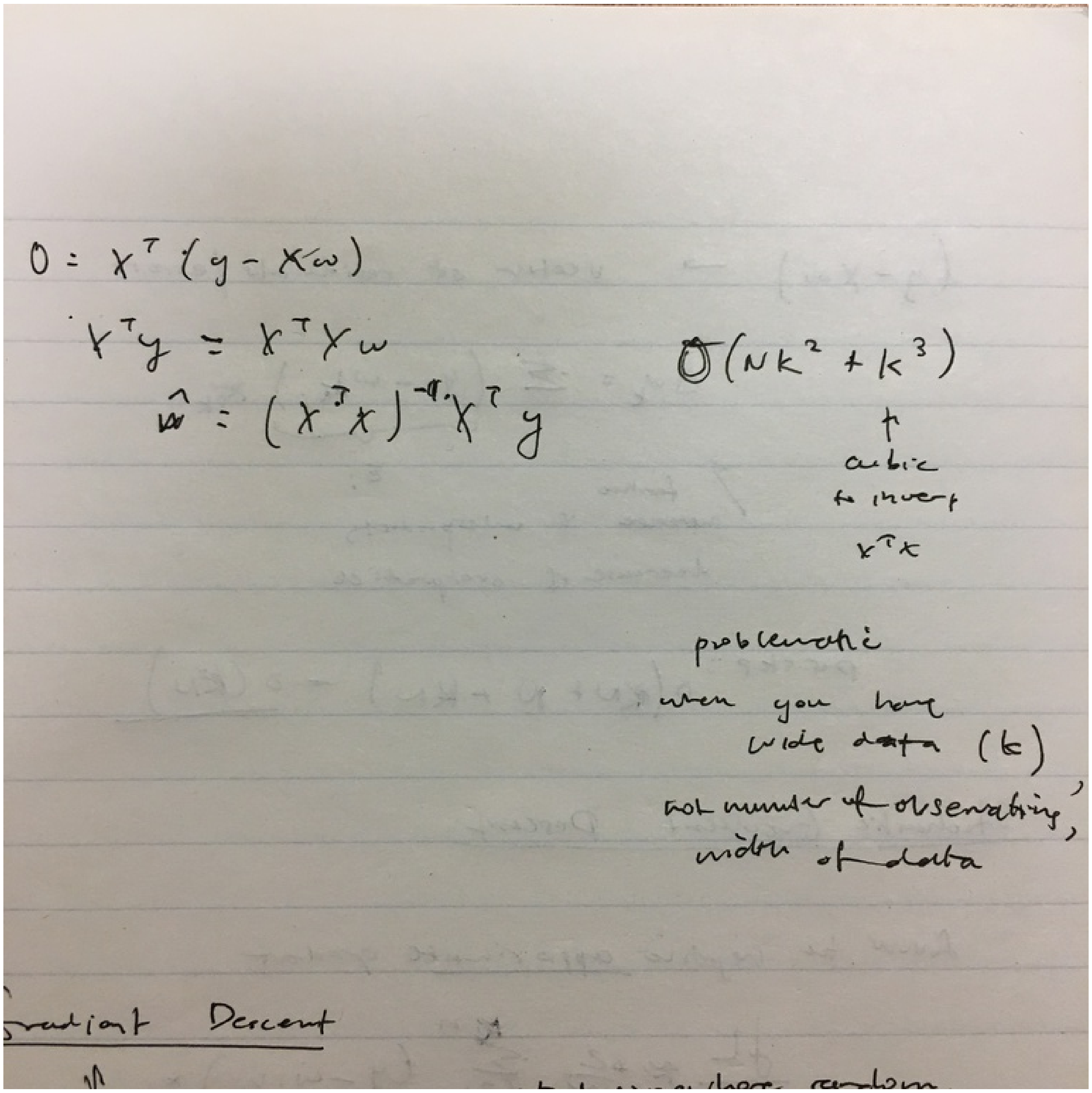

Figure 8: Since we are working with vectors, we can use linear algebra and matrix multiplication to solve for our coefficient vector w. Note, it becomes problematic when we have wide data (high dimensions k) as matrix operations are extremely expensive in this manner - number of observations is less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

Figure 9 Gradient Descent

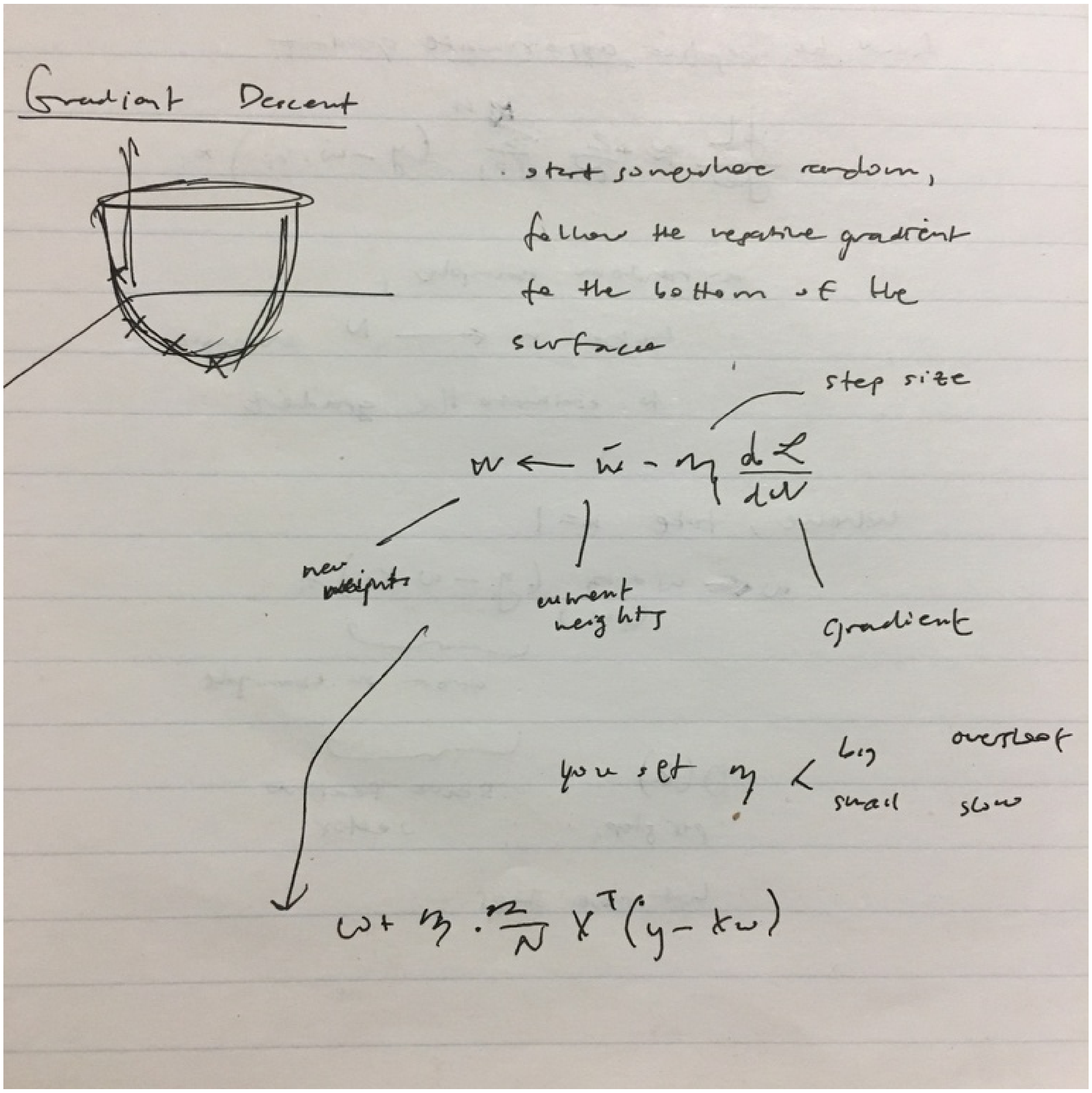

Figure 9: Start somewhere random, follow the negative gradient to the bottom of the surface. You set the learning rate (can be variable) - too big and it will overshoot the minimum, too small and gradient descent will take a long time

Figure 10 Vector of residuals/errors

$(y - Xw)$ → 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: $O(KN + N + KN) \rightarrow \underline{O(KN)}$

<u>Stochastic Gradient Descent</u>

follow the negative <u>approximate</u> gradient

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

Figure 11 Stochastic Gradient Descent

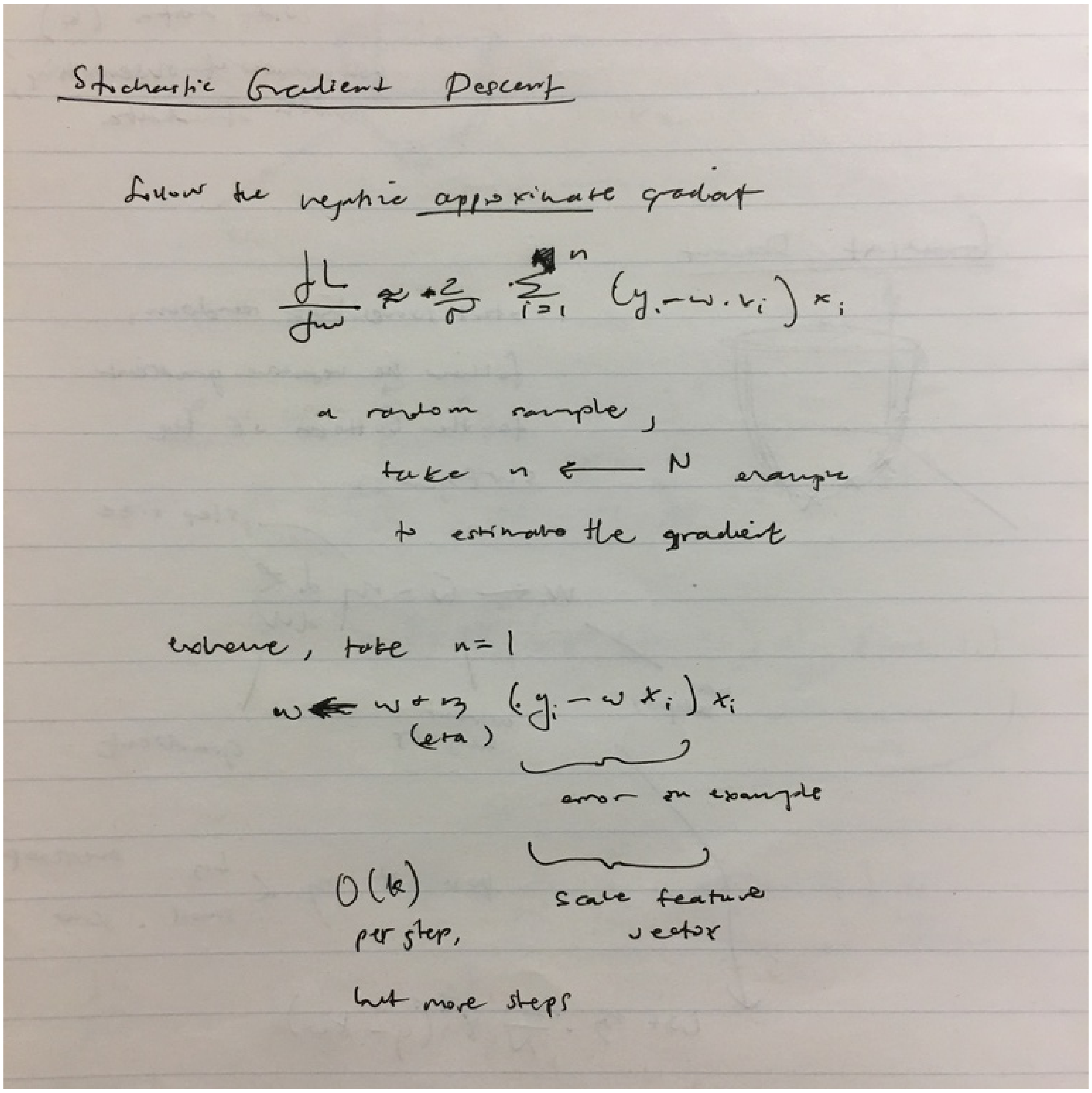

Figure 11: Follow the negative approximate gradient - take an n random sample to estimate the gradient