Machine Learning, Spring 2020

Regression

Reading Assignment: Chapter 3 & 4

Python tutorial: http://learnpython.org/

TensorFlow tutorial: https://www.tensorflow.org/tutorials/

PyTorch tutorial: https://pytorch.org/tutorials/

Prof. Yi Fang (yfang@nyu.edu)

Regression model: Overview

Regression model

1. A form of statistical modeling that attempts to evaluate the relationship between one variable (termed the dependent variable) and one or more other variables (termed the independent variables). It is a form of global analysis as it only produces a single equation for the relationship.
2. Regression model estimates the nature of the relationship between the independent and dependent variables.
   - Change in dependent variables that results from changes in independent variables, ie. size of the relationship.
   - Strength of the relationship.
   - Statistical significance of the relationship.

Examples

- Dependent variable is employment income – independent variables might be hours of work, education, occupation, sex, age, region, years of experience, unionization status, etc.

- Price of a product and quantity produced or sold:
  - Quantity sold affected by price. Dependent variable is quantity of product sold – independent variable is price.
  - Price affected by quantity offered for sale. Dependent variable is price – independent variable is quantity sold

Regression Model: Linear regression

Linear Regression

- Linear regression is a simple approach to supervised learning. It assumes that the dependence of Y on X_1, X_2, …,X_p is linear.
- Linear regression is simple but useful both conceptually and practically.

  - Attempt to determine causes of phenomena.
  - Prediction and forecasting of sales, economic growth, etc.

Linear Regression

- *x* is the independent variable; *y* is the dependent variable
- The regression model is

$$y = \beta_0 + \beta_1 x + \varepsilon$$

- The relationship between *x* and *y* is a linear or straight line relationship.
- Two parameters to estimate – the slope of the line β_1 and the *y*-intercept β_0 (where the line crosses the vertical axis).
- ε is the unexplained, random, or error component.

Linear Regression

- The regression model is $y = \beta_0 + \beta_1 x + \varepsilon$
- Data about *x* and *y* are obtained from a sample.
- From the sample of values of *x* and *y*, estimates b_0 of β_0 and b_1 of β_1 are obtained using the least squares or another method.
- The resulting estimate of the model is

$$\hat{y} = b_0 + b_1 x$$

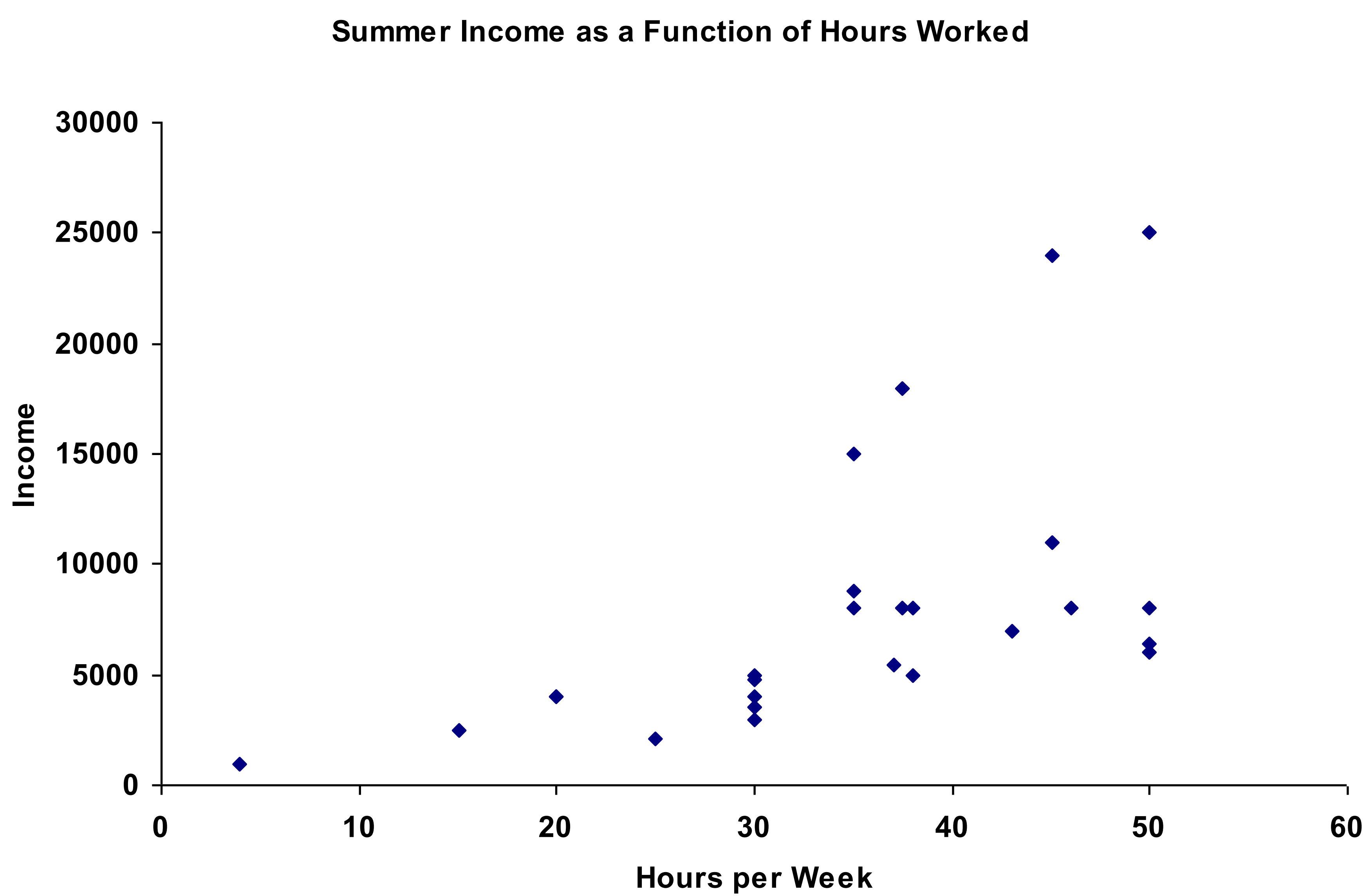
- The symbol $\hat{y}$ is termed "*y* hat" and refers to the predicted values of the dependent variable *y* that are associated with values of *x*, given the linear model.

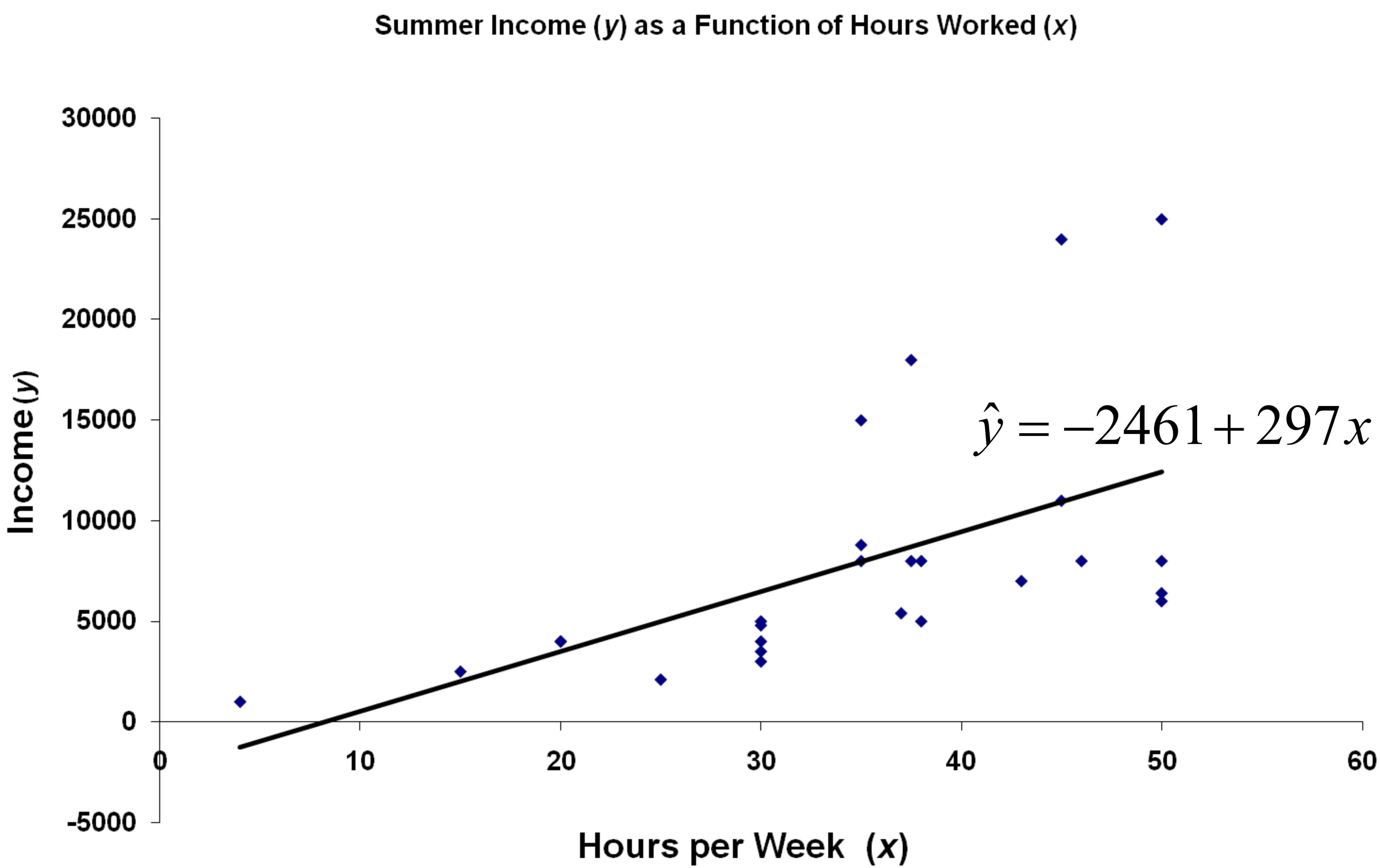
Example:

lent Variables

Independent Variables

Income	hrs/week	Income	hrs/week
8000	38	8000	35
6400	50	18000	37.5
2500	15	5400	37
3000	30	15000	35
6000	50	3500	30
5000	38	24000	45
8000	50	1000	4
4000	20	8000	37.5
11000	45	2100	25
25000	50	8000	46
4000	20	4000	30
8800	35	1000	200
5000	30	2000	200
7000	43	4800	30

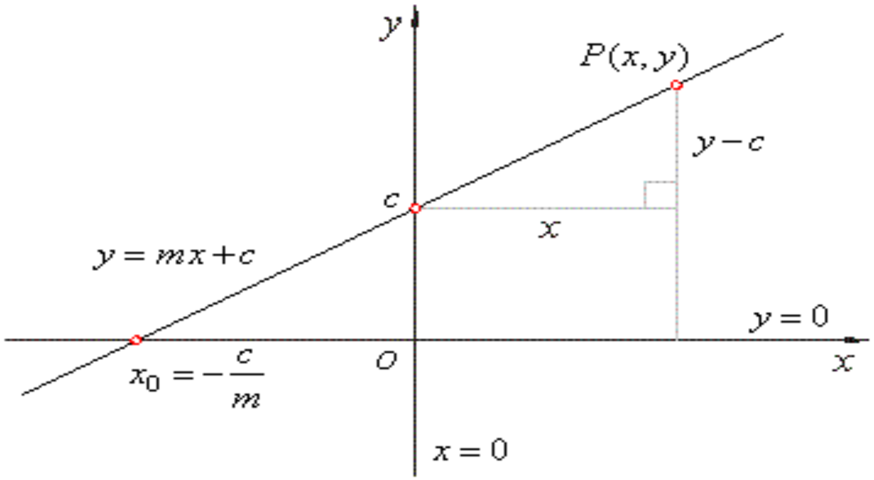

Summer Income as a Function of Hours Worked
Income
30000
25000
20000
15000
10000
5000
0
0
10
20
30
40
50
60
Hours per Week


Summer Income (y) as a Function of Hours Worked (x)
Income (y)
Hours per Week (x)
30000
25000
20000
15000
10000
5000
0
-5000
0
10
20
30
40
50
60
$\hat{y} = -2461 + 297x$

Training Linear Regression Model

- Model:

$$Y = mX + c$$



Source: http://www.nabla.hr/SlopeInterceptLineEqu.gif

Loss Function

- Loss is defined as the difference between the ground truth (actual values) and the predicted value
- For example, mean squared errors (MSE) loss

Mean Squared Error

$$E = \frac{1}{n} \sum_{i=0}^{n} (y_i - \bar{y}_i)^2$$

Mean Squared Error Equation

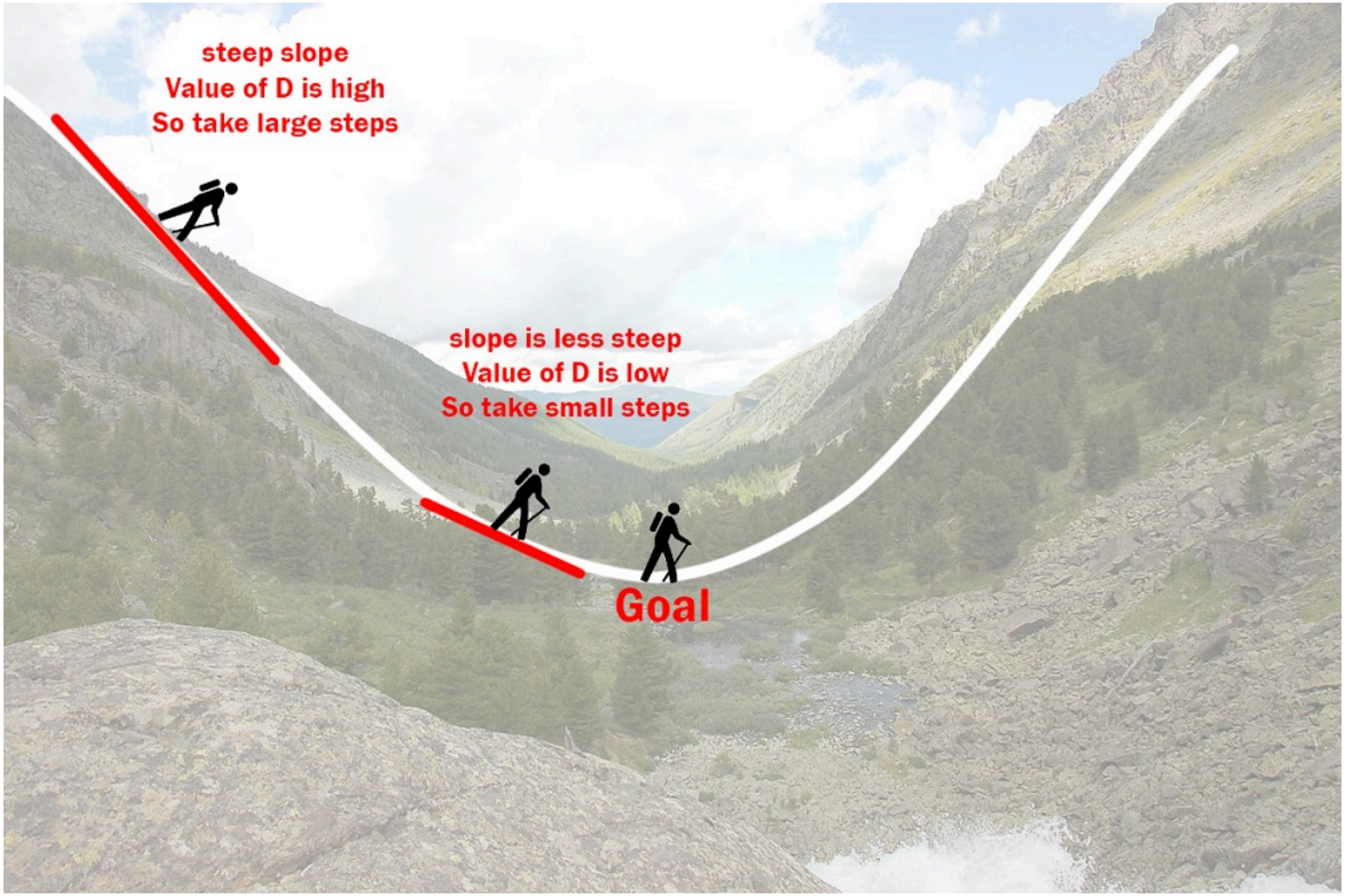
$$E = \frac{1}{n} \sum_{i=0}^{n} (y_i - (mx_i + c))^2$$

Substituting the value of $\bar{y}_i$

Training Method: Gradient Descent

- Gradient descent algorithm's main objective is to minimize the cost function. It is one of the best optimization algorithms to minimize errors (difference of actual value and predicted value).

Gradient Descent



- Partial derivative

$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

$$D_m = \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

Derivative with respect to **m**

- Partial Derivative

$$D_c = \frac{-2}{n} \sum_{i=0}^{n} (y_i - \bar{y}_i)$$

Derivative with respect to **c**

- Update parameter

$$m = m - L \times D_m$$

$$c = c - L \times D_c$$

Python Function

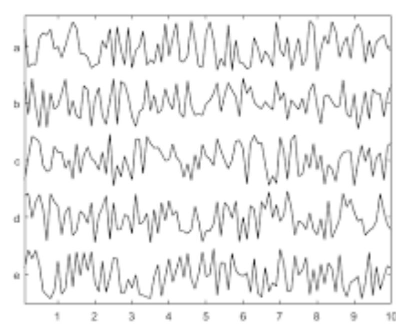
```
# Performing Gradient Descent
for i in range(epochs):
    Y_pred = m*X + c  # The current predicted value of Y
    D_m = (-2/n) * sum(X * (Y - Y_pred))  # Derivative wrt m
    D_c = (-2/n) * sum(Y - Y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c

print (m, c)
```

More General Form

$$\theta^T \mathbf{x} = \sum_{i=1}^{n} \theta_i x_i = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

weighted sum

Linear Regression: $Y = \theta^T \mathbf{x}$



Regression Model: Logistic regression

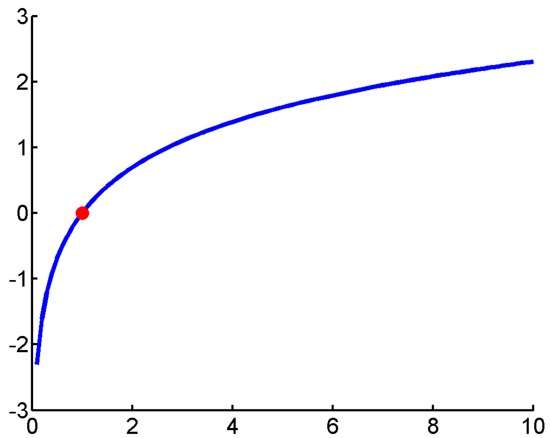
Logistic Regression

- Regression used to fit a curve to data in which the dependent variable is binary.
- Given some event with probability *p* of being 1, the **odds** of that event are given by:

$$\boxed{\text{odds} = p \,/\, (1-p)}$$

- The logit is the natural log of the odds
  logit(p) = ln(odds) = ln (p/(1-p))

Logit Transform



Logistic Regression

- In logistic regression, we seek a model:

$$\text{logit}(p) = \beta_0 + \beta_1 X$$

- That is, the log odds (logit) is assumed to be linearly related to the independent variable X
- So, now we can focus on solving an ordinary (linear) regression.

Logistic Regression-Recovering Probabilities

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X$$

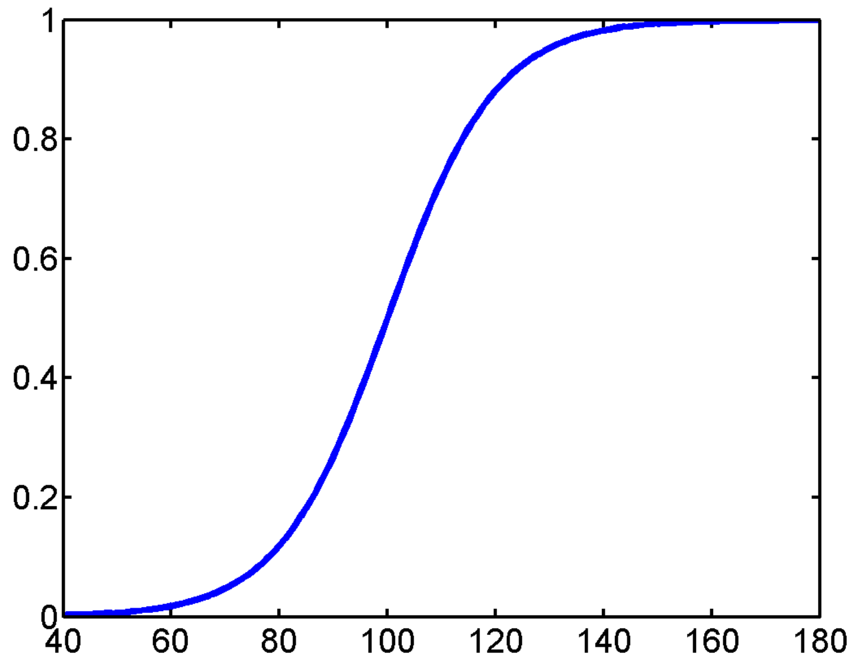
$$\Leftrightarrow \frac{p}{1-p} = e^{\beta_0 + \beta_1 X}$$

$$\Leftrightarrow p = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

which gives *p* as a sigmoid function!

Logistic Regression-Logistic Response Function

- When the response variable is binary, the shape of the response function is often sigmoidal:



Example Logistic Regression

- One application: The clinical research is interested in predicting response to treatment, where we might code survivors as 1 and those who don’t survive as 0

Typical application: Linear regression results

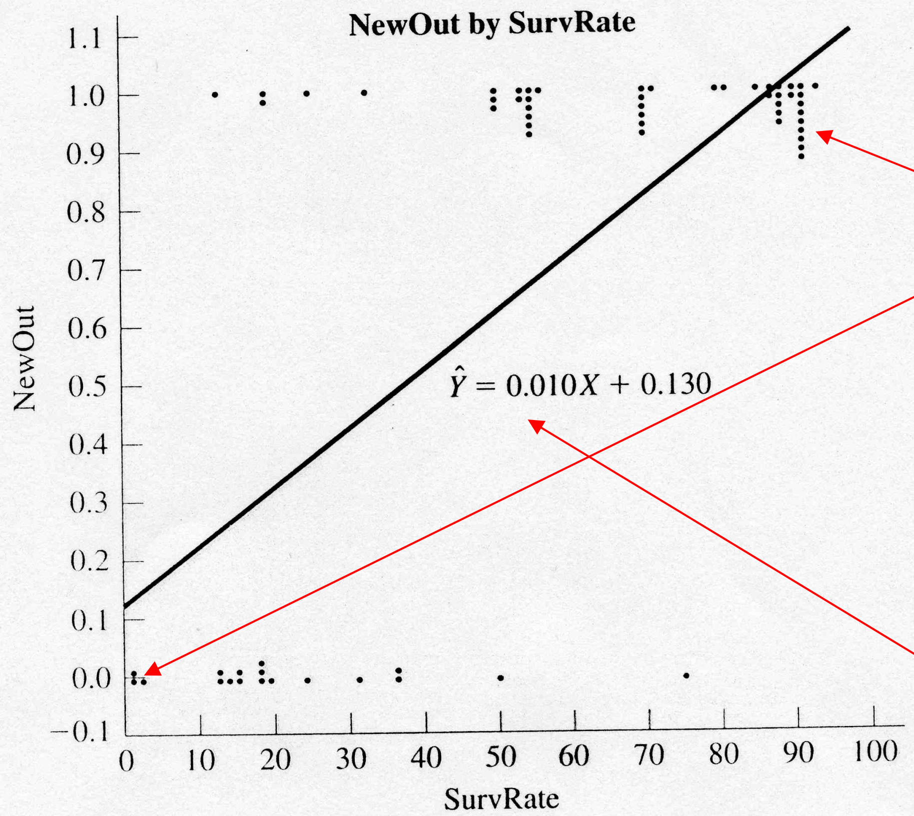


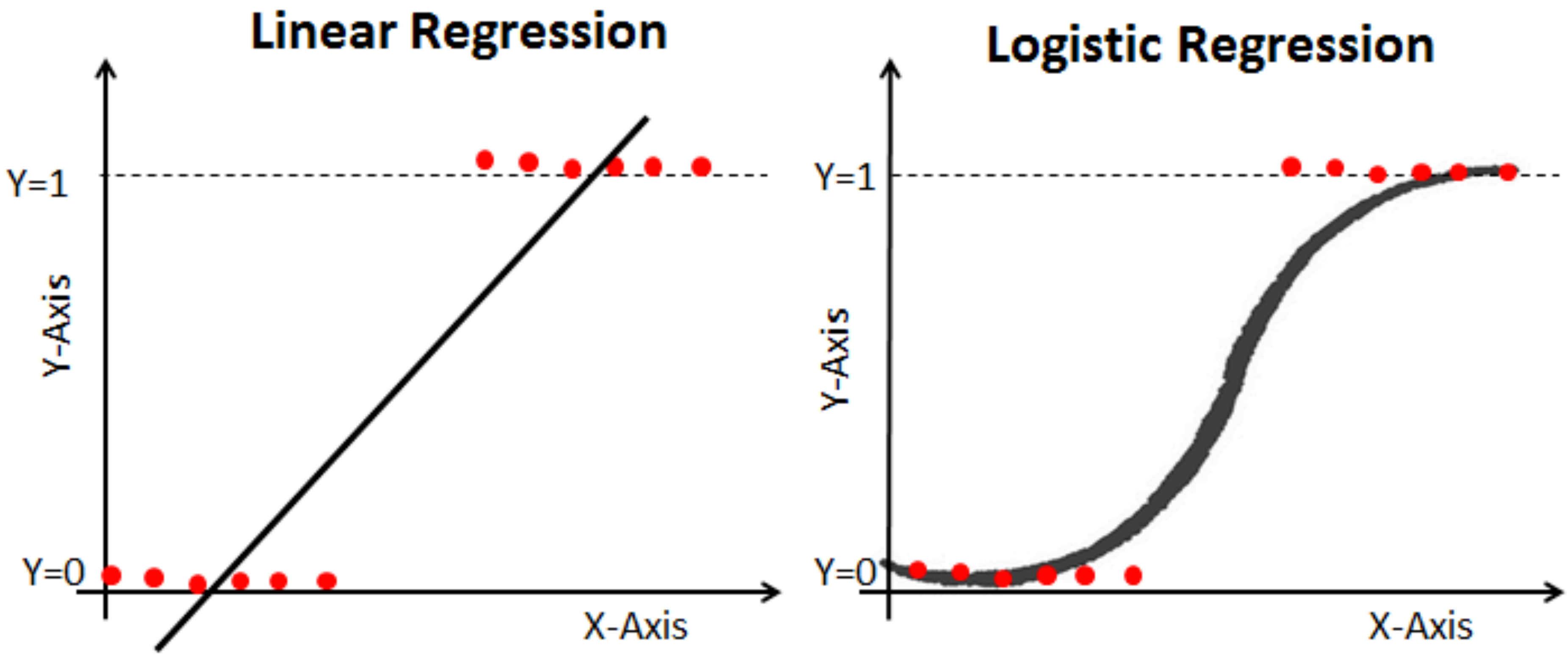
FIGURE 15.7 *Outcome as a function of SurvRate*

Observations:
For each value of SurvRate, the number of dots is the number of patients with that value of NewOut

Regression:
Standard linear regression

Problem: extending the regression line a few units left or right along the X axis produces predicted probabilities that fall outside of [0,1]

Comparison:


Linear Regression
Y=1
Y-Axis
Y=0
X-Axis
Logistic Regression
Y=1
Y-Axis
Y=0
X-Axis

Typical application: Medicine- A Better Solution

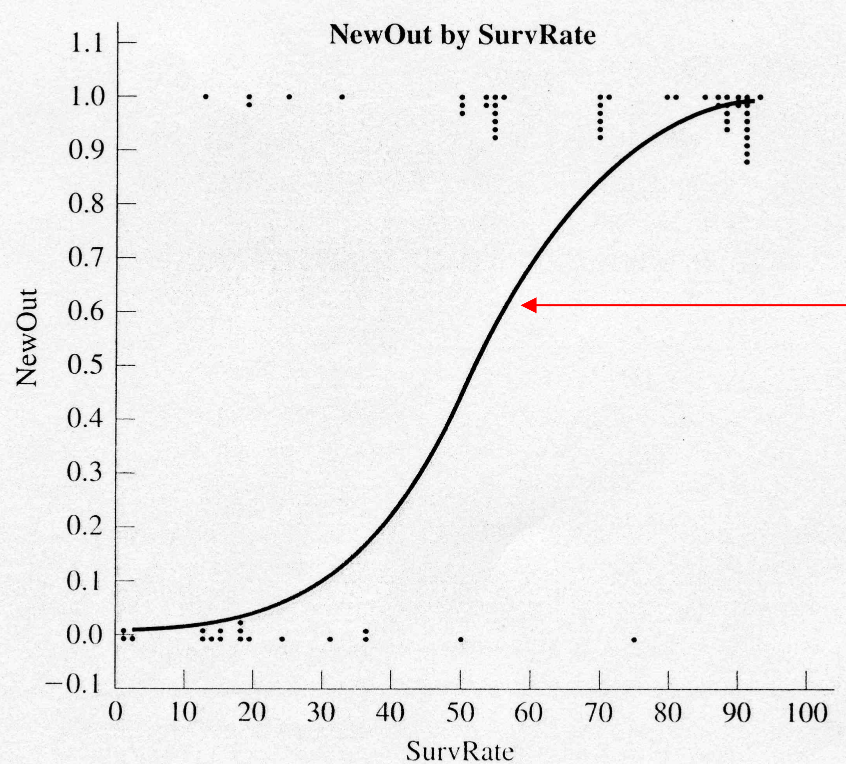


FIGURE 15.8 *More appropriate regression line for predicting outcome*

Regression Curve: Sigmoid function!

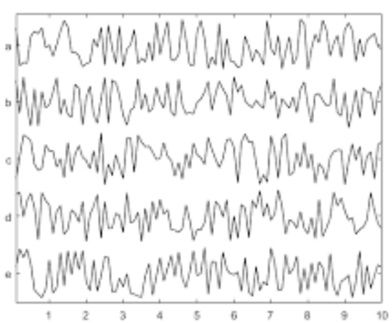
(bounded by asymptotes *y*=0 and *y*=1)

More General Form

$$\theta^T \mathbf{x} = \sum_{i=1}^{n} \theta_i x_i = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

weighted sum

Linear Regression: $Y = \theta^T \mathbf{x}$



Introduce:

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \text{sigmoid function}$$

Then:

Logistic Regression is a classification algorithm (I know, terrible name) that works by trying to learn a function that approximates $P(Y|X)$. It makes the central assumption that $P(Y|X)$ can be approximated as a sigmoid function applied to a linear combination of input features. Mathematically, for a single training datapoint $(\mathbf{x}, y)$ Logistic Regression assumes:

$$P(Y=1|\mathbf{X}=\mathbf{x}) = \sigma(z) \text{ where } z = \theta_0 + \sum_{i=1}^{m} \theta_i x_i$$

This assumption is often written in the equivalent forms:

$$P(Y=1|\mathbf{X}=\mathbf{x}) = \sigma(\theta^T \mathbf{x}) \qquad \text{where we always set } x_0 \text{ to be 1}$$

$$P(Y=0|\mathbf{X}=\mathbf{x}) = 1 - \sigma(\theta^T \mathbf{x}) \qquad \text{by total law of probability}$$

Training Logistic Regression Model

- Model:

$$P(Y=1|\mathbf{X}=\mathbf{x}) = \sigma(\theta^T\mathbf{x})$$

$$P(Y=0|\mathbf{X}=\mathbf{x}) = 1-\sigma(\theta^T\mathbf{x})$$

Loss Function

- Loss is defined as the difference between the ground truth (actual values) and the predicted value
- So how we define loss function for logistic regression?

Likelihood Loss

$$LL(\theta) =$$

$$-\sum_{i=0}^{n} y^{(i)} \log \sigma(\theta^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log[1 - \sigma(\theta^T \mathbf{x}^{(i)})]$$

Gradient of Loss Function

$$LL(\theta) = -\sum_{i=0}^{n} y^{(i)} \log \sigma(\theta^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log[1 - \sigma(\theta^T \mathbf{x}^{(i)})]$$

Partial Derivative:

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i=0}^{n} \left[y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)}) \right] x_j^{(i)}$$

How

Loss Function:

$$L(\theta) = \prod_{i=1}^{n} P(Y = y^{(i)} | X = \mathbf{x}^{(i)})$$

$$= \prod_{i=1}^{n} \sigma(\theta^T \mathbf{x}^{(i)})^{y^{(i)}} \cdot \left[1 - \sigma(\theta^T \mathbf{x}^{(i)})\right]^{(1-y^{(i)})}$$

Derivative of sigma:

$$\frac{\partial}{\partial z} \sigma(z) = \sigma(z)[1 - \sigma(z)]$$

Then:

Derivative of gradient for one datapoint $(\mathbf{x}, y)$:

$$\begin{aligned}
\frac{\partial LL(\theta)}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j} y \log \sigma(\theta^T \mathbf{x}) + \frac{\partial}{\partial \theta_j}(1-y)\log[1-\sigma(\theta^T \mathbf{x}] && \text{derivative of sum of terms} \\
&= \left[\frac{y}{\sigma(\theta^T x)} - \frac{1-y}{1-\sigma(\theta^T x)}\right] \frac{\partial}{\partial \theta_j}\sigma(\theta^T x) && \text{derivative of log } f(x) \\
&= \left[\frac{y}{\sigma(\theta^T x)} - \frac{1-y}{1-\sigma(\theta^T x)}\right] \sigma(\theta^T x)[1-\sigma(\theta^T x)]x_j && \text{chain rule + derivative of sigma} \\
&= \left[\frac{y-\sigma(\theta^T x)}{\sigma(\theta^T x)[1-\sigma(\theta^T x)]}\right] \sigma(\theta^T x)[1-\sigma(\theta^T x)]x_j && \text{algebraic manipulation} \\
&= \left[y-\sigma(\theta^T x)\right] x_j && \text{cancelling terms}
\end{aligned}$$

Training Method: Gradient Descent

- Gradient descent algorithm's main objective is to minimize the cost function. It is one of the best optimization algorithms to minimize errors (difference of actual value and predicted value).

Updating rule:

$$\theta_j^{\text{new}} = \theta_j^{\text{old}} + \eta \cdot \frac{\partial LL(\theta^{\text{old}})}{\partial \theta_j^{\text{old}}}$$

$$= \theta_j^{\text{old}} + \eta \cdot \sum_{i=0}^{n} \left[y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)}) \right] x_j^{(i)}$$

Python Function

```
def sigmoid(x):
    # Activation function used to map any real value between 0 and 1
    return 1 / (1 + np.exp(-x))

def net_input(theta, x):
    # Computes the weighted sum of inputs
    return np.dot(x, theta)

def probability(theta, x):
    # Returns the probability after passing through sigmoid
    return sigmoid(net_input(theta, x))
```

Next, we define the `cost` and the `gradient` function.

```
def cost_function(self, theta, x, y):
    # Computes the cost function for all the training samples
    m = x.shape[0]
    total_cost = -(1 / m) * np.sum(
        y * np.log(probability(theta, x)) + (1 - y) * np.log(
            1 - probability(theta, x)))
    return total_cost

def gradient(self, theta, x, y):
    # Computes the gradient of the cost function at the point theta
    m = x.shape[0]
    return (1 / m) * np.dot(x.T, sigmoid(net_input(theta,   x)) - y)
```

Fitting function:

Let's also define the `fit` function which will be used to find the model parameters that minimizes the cost function. In this blog, we coded the gradient descent approach to compute the model parameters. Here, we will use `fmin_tnc` function from the `scipy` library. It can be used to compute the minimum for any function. It takes arguments as

- func: the function to minimize
- x0: initial values for the parameters that we want to find
- fprime: gradient for the function defined by 'func'
- args: arguments that needs to be passed to the functions.

```
def fit(self, x, y, theta):
    opt_weights = fmin_tnc(func=cost_function, x0=theta,
                  fprime=gradient,args=(x, y.flatten()))
    return opt_weights[0]

parameters = fit(X, y, theta)
```

Reference

- https://ocw.mit.edu/courses/mathematics/18-05-introduction-to-probability-and-statistics-spring-2014/class-slides/MIT18_05S14_class25slides.pdf
- http://finance.wharton.upenn.edu/~mrrobert/resources/Teaching/CorpFinPhD/Linear-Regression-Slides.pdf
- http://www-hsc.usc.edu/~eckel/biostat2/slides/lecture13.pdf
- https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931

- We have used huge amount of online resources for this course. All of them are the sole copyright holders of their material. Here we have referenced them with proper credits.

- https://towardsdatascience.com/building-a-logistic-regression-in-python-301d27367c24