Linear Regression

Many slides are adapted from Dr. Michael DeGiorgio's lecture slides

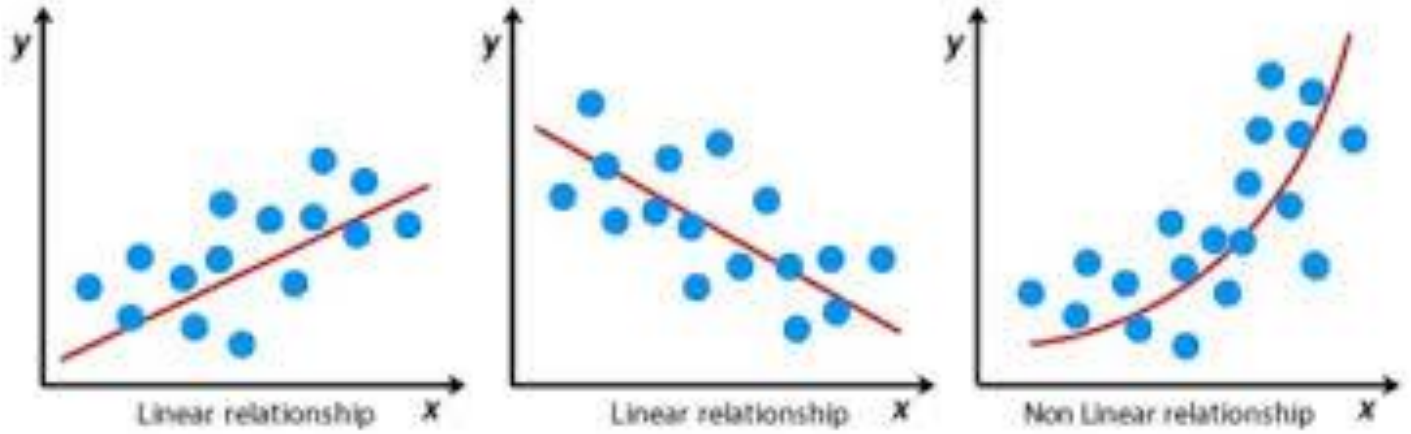
Linear Regression

- **Linear regression** is a parametric (model-based) supervised learning method that assumes a response Y can be estimated well by a linear combination of input features $X = [X_1, X_2, \dots, X_p]$.

- That is, linear regression assumes $Y = f(X_1, X_2, \dots, X_p)$ has the form

$$Y = w_0 + w_1 X_1 + w_2 X_2 + \cdots + w_p X_p$$

where $w = [w_0, w_1, \dots, w_p]$ is the collection of model **parameters**, which are also known as **regression coefficients**.



Linear Regression

- The goal is that, given some training data, we hope to be able to learn a linear relationship between Y and input features $X = [X_1, X_2, \dots, X_p]$ so that given measurements of these features on a new observation, we may be able to accurately predict its response Y.

- We predict the value of the outcome variable Y from the predictor(s) X

Simple Linear Regression: $Y_i = w_0 + w_1 X_1$

Multiple Linear Regression: $Y_i = w_0 + \sum_{i=1}^{p} w_i X_i$

Simple Linear Regression

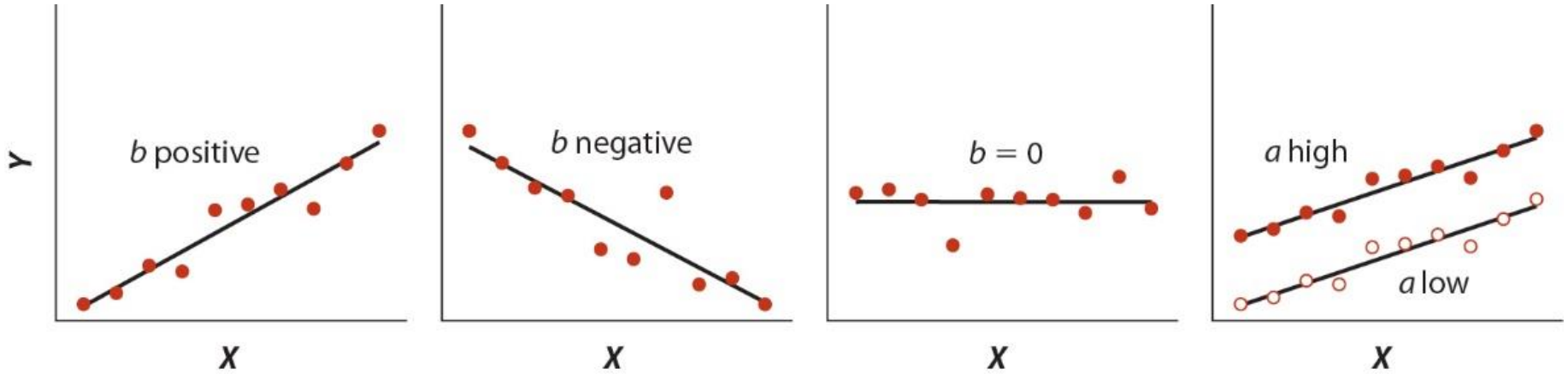
- **Simple linear regression** predicts response Y from *one* feature X as

$$Y = w_0 + w_1 X$$

which is just an equation for a line with intercept w_0 and slope w_1.

- Here, w_0 is the value of Y when $X = 0$ and w_1 describes the rate of change in Y as a function of X.

- The w_0 and w_1 parameters are unknown quantities from a **population** that we estimated from our training data **sample**. $b = w_1$ & $a = w_0$



Simple Linear Regression

- We first fit the simple linear regression model

$$Y = w_0 + w_1 X$$

to the sampled training data (x_i, y_i), $i = 1,2, \ldots, N$ to estimate the population parameters w_0 and w_1.

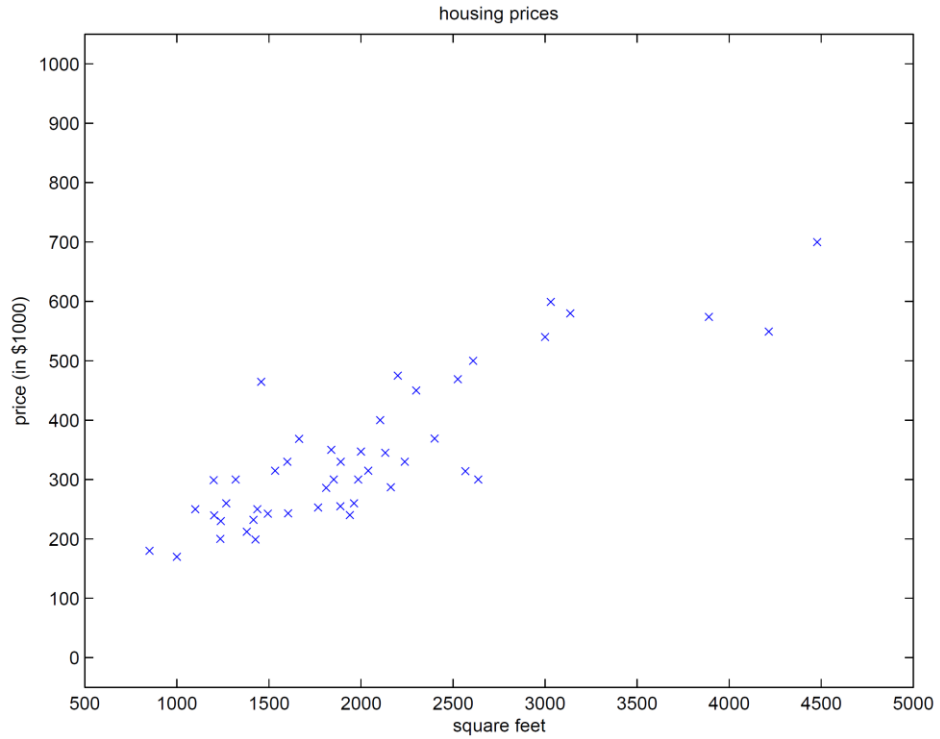
- The estimates $\hat{w}_0$ and $\hat{w}_1$ of the parameters provides us with a relationship between Y and X that we can use to make predictions.

- Given a new observation X, the hope is that our model $\hat{f}(X)$ (or $\hat{Y}$) can accurately predict the response Y with a model of the form

$$\hat{Y} = \hat{w}_0 + \hat{w}_1 X$$

Example on estimating housing prices

- Consider the following set of plotted datapoints, demonstrating the relationship between price of a house (y-axis) and square footage of the home (x-axis).



- Price appears linearly related with square footage, and we wish to uncover what this linear relationship is.

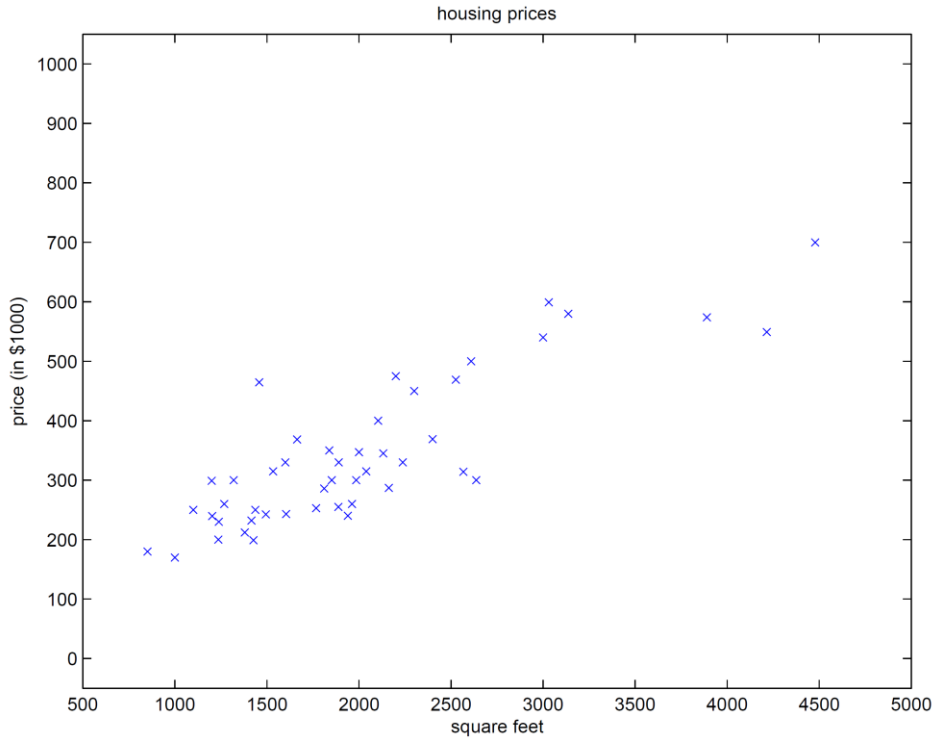
http://cs229.stanford.edu/notes/cs229-notes1.pdf

Example on estimating housing prices

- A linear model relating housing price (Y) to square footage (X) would take the following form

$$Y = w_0 + w_1 X$$

- Our goal is to estimate w_0 and w_1 that best fit the data.



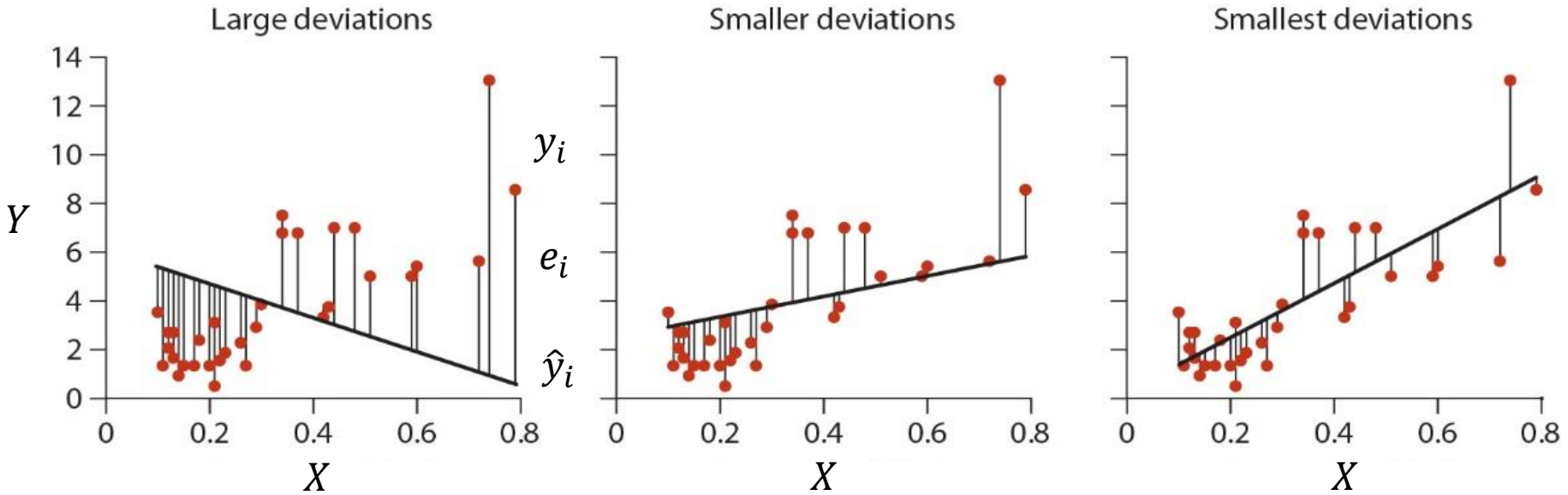
http://cs229.stanford.edu/notes/cs229-notes1.pdf

How to find the best-fit regression line?

- Finding best-fit regression line means correctly estimating the parameters w_0 and w_1
- To estimate the parameters w_0 and w_1, we need to minimize the deviations/error (differences) between observed and predicted responses.
- That is, for training observation (x_i, y_i), $i = 1,2, \ldots, N$, let the prediction of its response y_i based on input features x_i be

$$\hat{y}_i = \hat{w}_0 + \hat{w}_1 x_i$$

with the **residual** as $e_i = y_i - \hat{y}_i$.

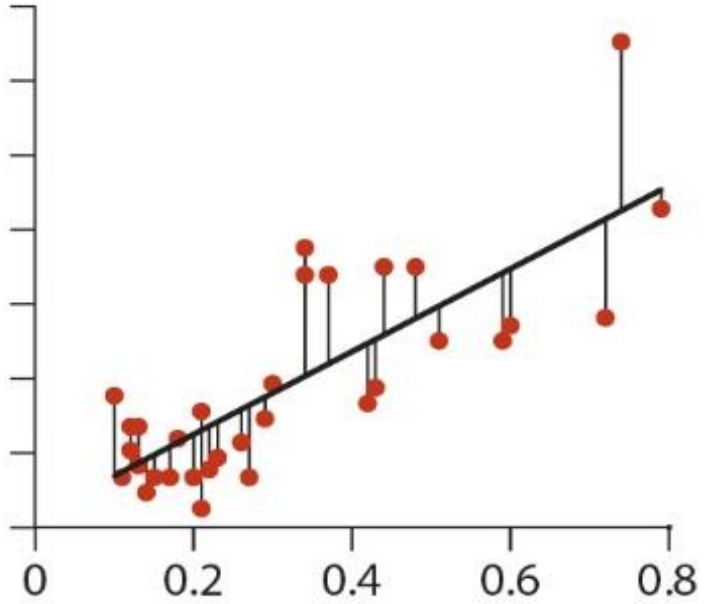


The cost/error function $J(w)$

- The residual sum of squares (RSS) quantifies the sum of squared residuals as

$$\text{RSS} = e_1^2 + e_2^2 + \cdots + e_N^2$$

  - Each residual e_i is the error of data x_i
  - This cost/error formula is called squared sum of errors



- In machine learning, a **cost/error function** quantifies the amount of error between predictions and observations(target values), and we therefore seek to minimize the least squares regression cost function

$$J(w) = \text{RSS} = \sum_i (y_i - \hat{y}_i)^2 = e_1^2 + e_2^2 + \cdots + e_N^2$$

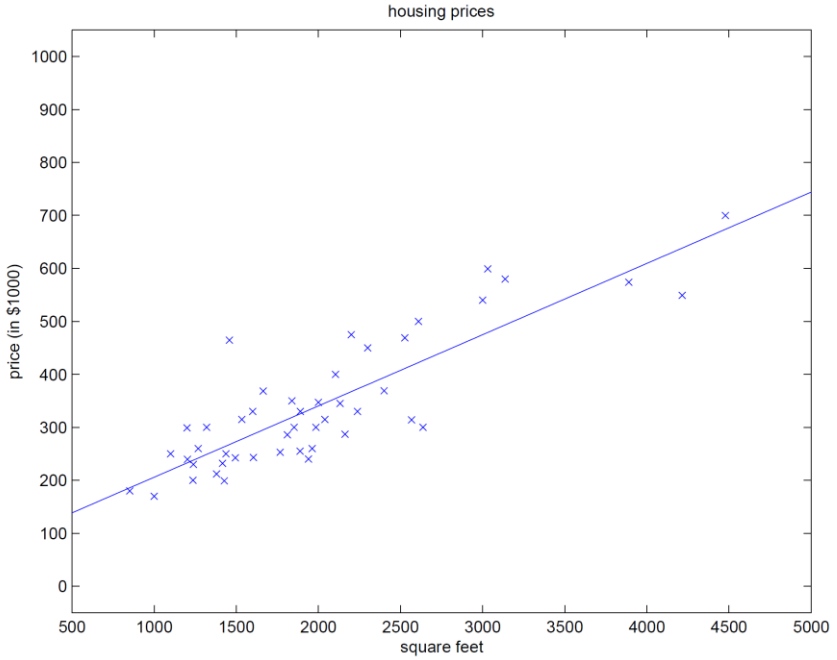
- Therefore, the best fit line in linear regression is the one that minimizes RSS

Example on estimating housing prices

- This best-fit model is shown as the blue line in the center of the datapoints in the graph below.

- The best-fit parameters are $\widehat{w}_0 = 71.27$ and $\widehat{w}_1 = 0.1345$, yielding

$$Y = 71.27 + 0.1345X$$

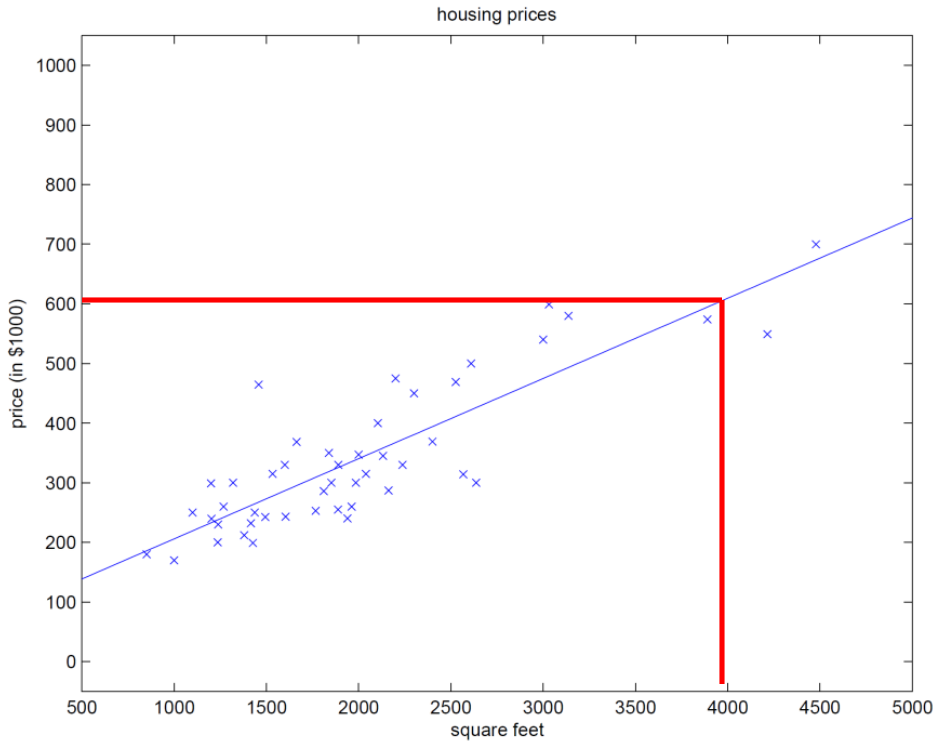


http://cs229.stanford.edu/notes/cs229-notes1.pdf

Example on estimating housing prices

- We can now predict Y from a new observation (*e.g.*, $X = 4000$) as

$$\hat{Y} = \hat{w}_0 + \hat{w}_1 X$$
$$= 71.27 + 0.1345(4000)$$
$$= 609.27$$



http://cs229.stanford.edu/notes/cs229-notes1.pdf

Extending the housing price example

- This housing price example was overly simplistic, as it is likely that we will have access to measurements on other characteristics of a home that could lead to better prediction.

- As an example, online real-estate companies that provide predictions of home prices may use a number of additional features such as

  - Location
  - Numbers of bedrooms and bathrooms
  - Price of comparable recently-sold homes

  Etc.

- We want to ultimately be able to build models that incorporate as many features as possible, and that will automatically put greater emphasis on those that are important for prediction.

Multiple Linear Regression Model

- In general, for linear regression we assume that a response Y can be estimated by a linear combination of p features $X = [X_1, X_2, \dots, X_p]$ as

$$Y = w_0 + w_1 X_1 + w_2 X_2 + \cdots + w_p X_p$$

- We seek to find the $p + 1$ model parameters $\mathrm{w} = [w_0, w_1, \dots, w_p]$ that minimize the cost function $J(w) = \mathrm{RSS}$ to learn the relationship between Y and $X = [X_1, X_2, \dots, X_p]$.

- Our goal is that given measured features $X = [X_1, X_2, \dots, X_p]$ on a new observation, we wish to accurately predict the response Y for that observation.

Cost/Error Function of Linear Regression

- To estimate the regression coefficients, we will fit a linear model to a set of training data (x_i, y_i), $i = 1,2, \dots, N$, using **least squares**.
- That is, we will identify the coefficients w that minimize the residual sum of squares

$$J(w) = \text{RSS}(w) = \sum_{i=1}^{N} \left(y_i - x_i^T w\right)^2$$

which represents the sum of squared differences between the linear function $x_i^T w$ used to predict the output and the true output y_i. Here, $J(w)$ is known as our **cost/error/loss function**.

- For notational simplicity, we assume a dummy input $X_0 = 1$.
  Now $w_0 + \sum_{i=1}^{n} w_i X_i$ is represented as $w_0 X_0 + \sum_{i=1}^{n} w_i X_i = \sum_{i=0}^{n} w_i X_i = w^T x$

- $J(w)$ is a convex function, and so an identified local minimum will also be the global minimum.
- Two possible methods to solve this problem
  - matrix method (will explain in this chapter)
  - gradient method (will explain in logistic regression)

Writing the Cost Function in matrix notation

- We construct the matrix of inputs $\mathbf{X} \in \mathbb{R}^{N\times(p+1)}$ with the ith row being the input of the ith training observation, and the vector $\mathbf{y} \in \mathbb{R}^{N}$ with the ith element being the output of the ith training observation. (x_{ij} means jth feature in ith observation) That is (assume $x_{i0} = 1$)

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Np} \end{bmatrix} = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

- Note that $\mathbf{y} - \mathbf{X}w$ has value

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Np} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_p \end{bmatrix} = \begin{bmatrix} y_1 - x_1^T w \\ y_2 - x_2^T w \\ \vdots \\ y_N - x_N^T w \end{bmatrix}$$

- We can rewrite the cost function $J(w)$ in matrix notation as

$$J(w) = \text{RSS}(w) = \sum_{i=1}^{N} \left(y_i - x_i^T w\right)^2 = (\mathbf{y} - \mathbf{X}w)^T(\mathbf{y} - \mathbf{X}w)$$

(*) Finding w that minimizes $J(w)$

- Taking the partial derivative with respect to w_k, we have (assume $x_{i0} = 1$)

$$\frac{\partial J(w)}{\partial w_k} = -\sum_{i=1}^{N} 2x_{ik}\left(y_i - \underbrace{\sum_{j=0}^{p} x_{ij}w_j}_{x_i^T w}\right) = -2\sum_{i=1}^{N} x_{ik}(y_i - x_i^T w)$$

which can be rewritten as

$$\frac{\partial J(w)}{\partial w_k} = -2\begin{bmatrix} x_{1k} & x_{2k} & \cdots & x_{Nk} \end{bmatrix}\begin{bmatrix} y_1 - x_1^T w \\ y_2 - x_2^T w \\ \vdots \\ y_N - x_N^T w \end{bmatrix}$$

$$= -2\begin{bmatrix} x_{1k} & x_{2k} & \cdots & x_{Nk} \end{bmatrix}(\mathbf{y} - \mathbf{X}w)$$

where $\begin{bmatrix} x_{1k} & x_{2k} & \cdots & x_{Nk} \end{bmatrix}$ is an N-dimensional row vector.

(*) Finding w that minimizes $J(w)$

- Putting this together, to identify the minimum, we take the gradient with respect to w, which is a $(p+1)$-dimensional vector, with dimension k the partial derivative of $J(w)$ with respect to w_k, $k \in \{0,1,\dots,p\}$. The gradient is therefore

$$\frac{\partial J(w)}{\partial w} = -2\begin{bmatrix} [x_{10} & x_{20} & \cdots & x_{N0}](\mathbf{y}-\mathbf{X}w) \\ [x_{11} & x_{21} & \cdots & x_{N1}](\mathbf{y}-\mathbf{X}w) \\ & & \vdots & \\ [x_{1p} & x_{2p} & \cdots & x_{Np}](\mathbf{y}-\mathbf{X}w) \end{bmatrix}$$

$$= -2\begin{bmatrix} x_{10} & x_{20} & \cdots & x_{N0} \\ x_{11} & x_{21} & \cdots & x_{N1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1p} & x_{2p} & \cdots & x_{Np} \end{bmatrix}(\mathbf{y}-\mathbf{X}w)$$

$$= -2\mathbf{X}^T(\mathbf{y}-\mathbf{X}w)$$

Finding parameter w that minimizes $J(w)$

- Setting the gradient to the 0 vector, gives

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}w) = 0$$

which yields

$$\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{w} = 0$$

or

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y}$$

- If $\mathbf{X}^T\mathbf{X}$ is invertible, the estimated coefficients (**normal equations**) are

$$\widehat{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

and is the vector w $\in \mathbb{R}^{p+1}$ that minimizes

$$J(w) = \|\mathbf{y} - \mathbf{X}w\|_2^2$$

with $\|v\|_2 = \sqrt{v_1^2 + v_2^2 + \cdots v_n^2}$ for $v = [v_1, v_2, \ldots, v_n] \in \mathbb{R}^n$, and is known as the L_2 norm.

Concrete example

- Consider the set of $N = 3$ training observations at $p = 1$ inputs.

$$\{(x_1, y_1) = (1,1), (x_2, y_2) = (2,4), (x_3, y_3) = (3,4)\}$$

- Use least squares to fit the best-fit line to these training data. Based on the fit model, predict the output Y for input $X = 4$.

- The input matrix $\mathbf{X}$ and output vector $\mathbf{y}$ of the training data are

$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 4 \\ 4 \end{bmatrix}$$

- We want to find the vector $w = [w_0, w_1] = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$. We start by computing the transpose of the design matrix

$$\mathbf{X}^T = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix}$$

Concrete example

- We then compute $\mathbf{X}^T\mathbf{X}$ as

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}$$

- Taking the inverse, we obtain

$$\left(\mathbf{X}^T\mathbf{X}\right)^{-1} = \begin{bmatrix} 7/3 & -1 \\ -1 & 1/2 \end{bmatrix}$$

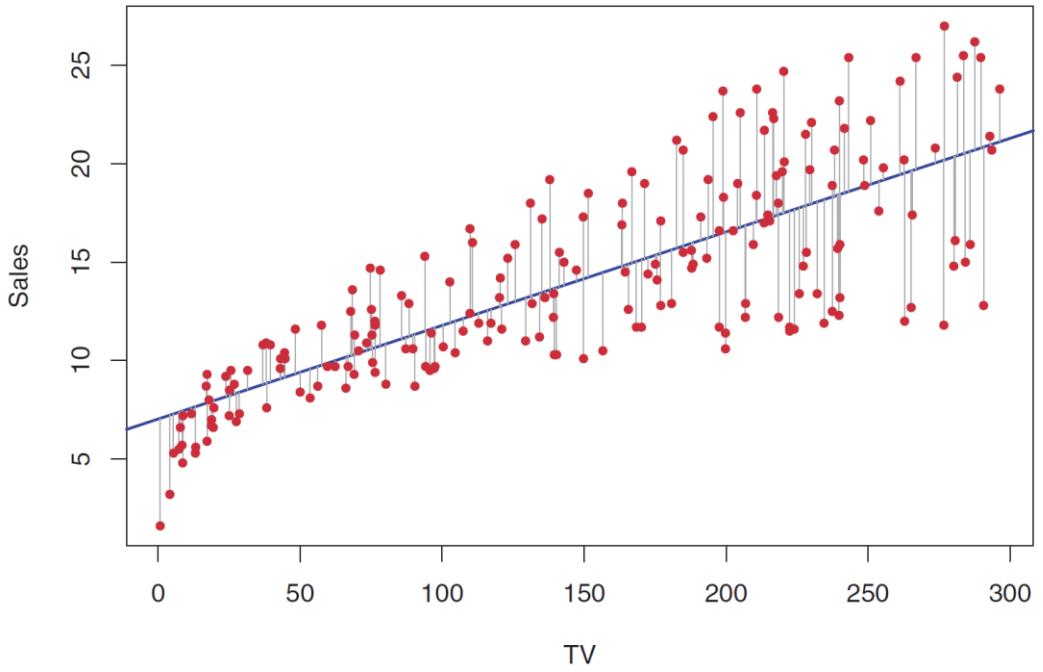
- An estimate of $\widehat{w}$ is

$$\widehat{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} = \begin{bmatrix} 7/3 & -1 \\ -1 & 1/2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 3/2 \end{bmatrix}$$

yielding $\widehat{w}_0 = 0$ and $\widehat{w}_1 = 3/2$. Therefore, the estimate of the output is

$$\widehat{Y} = \widehat{w}_0 + \widehat{w}_1 X = (3/2)X = (3/2) \cdot 4 = 6$$

Advertising data example



- Least squares fit for the regression of sales onto money spent on TV advertising.
- Gray segments are residuals (error), with model fit a compromise based on the average (or sum) of the squares of these residuals.
- When there are a lot of variables, computing inverse matrix takes too much time. Use gradient method instead. (explained in logistic regression chapter)

James *et al* (2017) *An Introduction to Statistical Learning: with Applications in R*. Springer.

Linear Regression in sklearn

```
import numpy as np
from sklearn.linear_model import LinearRegression

x = [[0, 1], [5, 1], [15, 2], [25, 5], [35, 11], [45, 15], [55, 34], [60, 35]]
y = [4, 5, 20, 14, 32, 22, 38, 43]
x, y = np.array(x), np.array(y)

print(x)
print(y)

model = LinearRegression()
model.fit(x, y)

r_sq = model.score(x, y)
print(f"intercept: {model.intercept_}")
print(f"slope: {model.coef_}")
print(f"coefficient of r-square: {r_sq}")
```

```
intercept: 5.5225
slope: [0.4470 0.2550]
coefficient of r-square: 0.8615
```

Logistic Regression

Prediction under a linear model

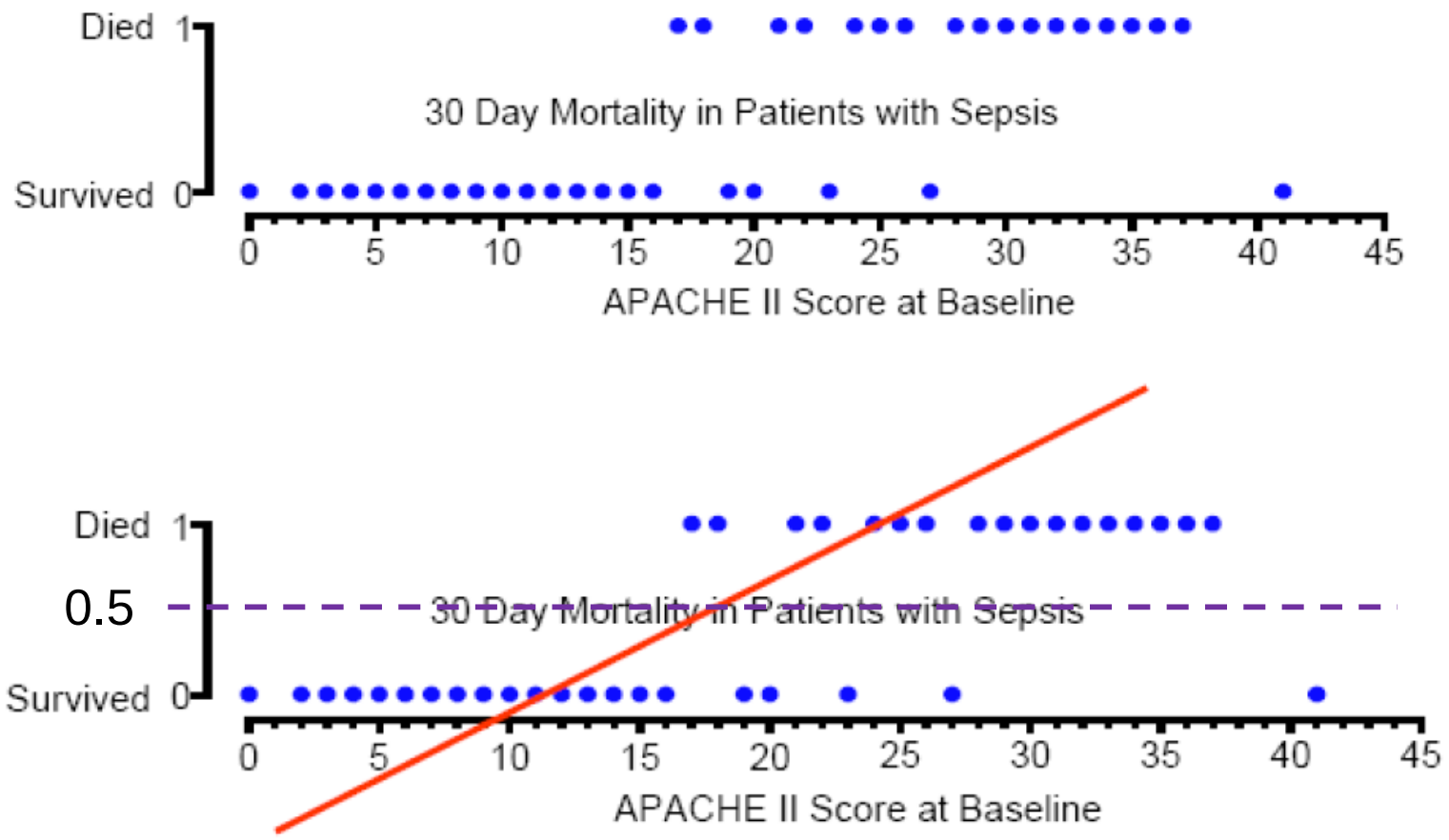
- We begin by developing a simple parametric approach for predicting output Y from given input $X = [X_1, X_2, \dots, X_p] \in \mathbb{R}^p$, where the vector of inputs X is based upon p features.

- Assuming that Y is linearly related to X, we can estimate (predict) the output Y assuming the linear model

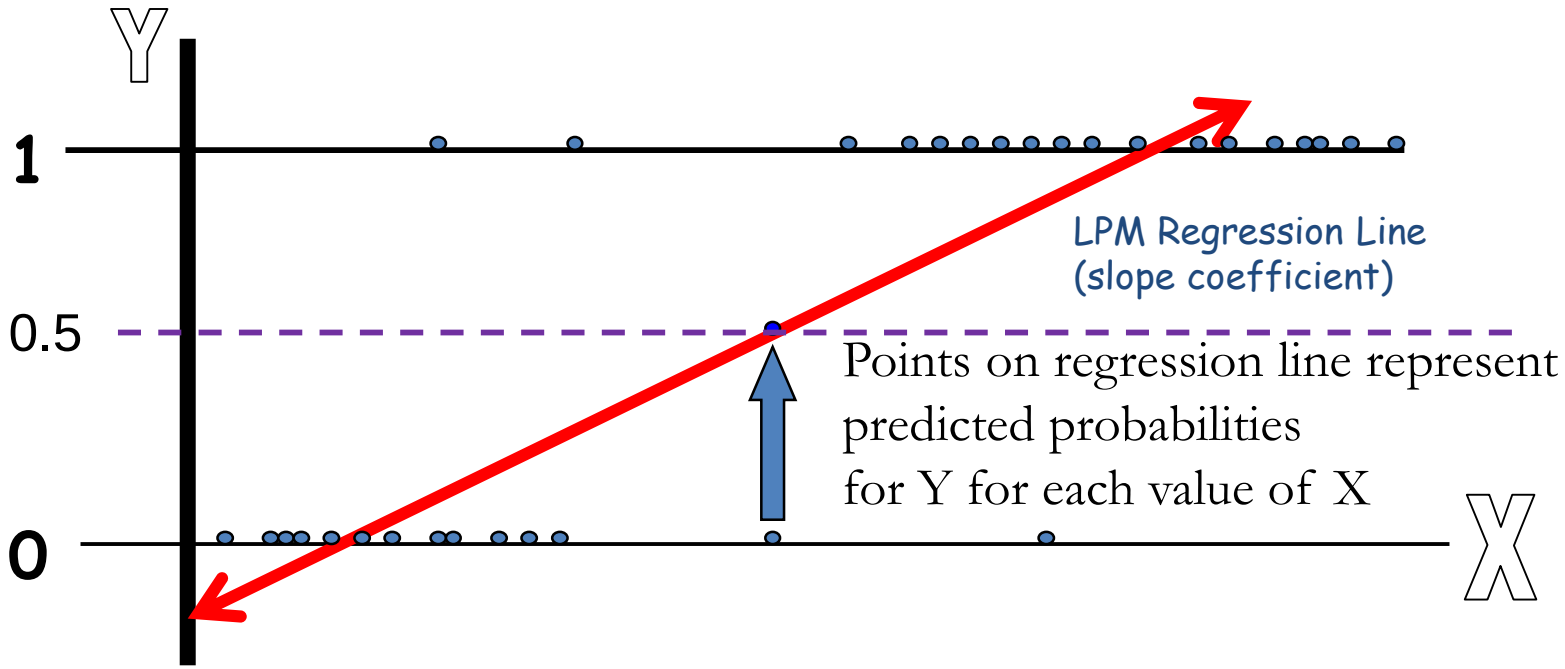
$$\hat{Y} = \hat{w}_0 + \sum_{j=1}^{p} X_j \hat{w}_j$$

where $\hat{w}_0$ is the intercept of our trained model, and $\hat{w}_j$ represents the rate of change of the output as a function of changes in feature j.

Linear Regression as Classifier



Linear Regression as Classifier



- Linear Regression as a Classifier
  Y=0 if p(X) < 0.5
  Y=1 if p(X) >=0.5

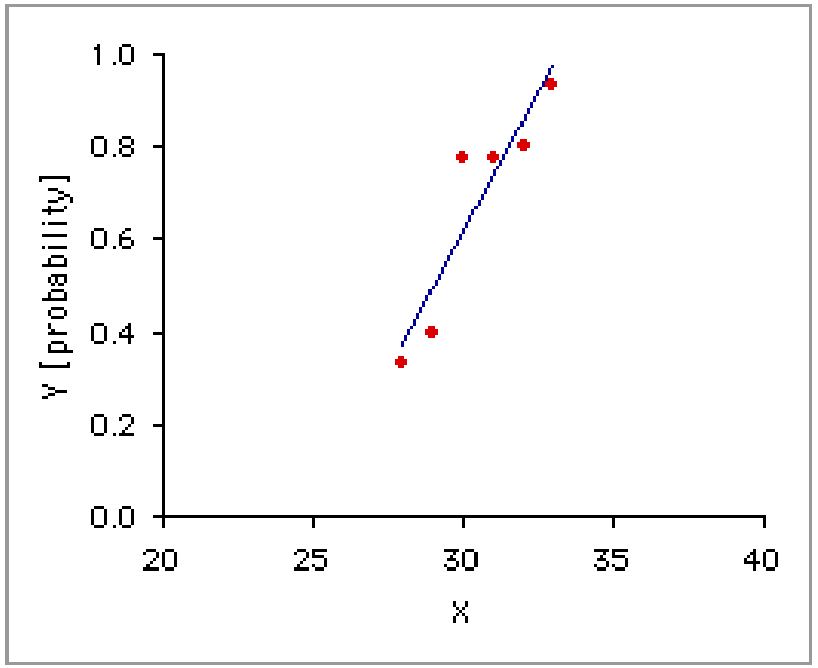
- However, if we use linear regression, some of our estimates might be **outside the [0, 1] interval,** which does not make sense as a probability

- Linear regression is also very vulnerable to outlier

Linear vs Logistic Regression

- How to make sure that the outcome will always be within that range [0,1].
- Transform the ouput using sigmoid(logistic) function

A. Ordinary Linear Regression

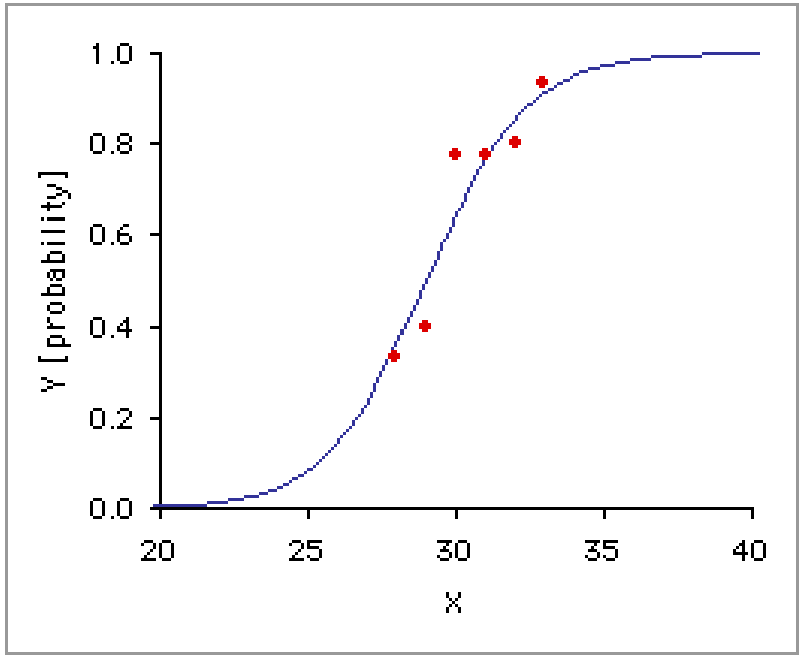


$$Y = w_0 X_0 + \sum_{i=1}^{n} w_i X_i$$

Squash using sigmoid function

$$Y = \frac{1}{1 + e^{-X}}$$

B. Logistic Regression



$$Y = \frac{1}{1 + e^{-(w_0 X_0 + \sum_{i=1}^{n} w_i X_i)}}$$

Logistic Regression

- Let x be the data instance, and y be the class label
- In logistic regression, we learn the conditional distribution $p(y|x)$ directly
- Let $w = [w_0, w_1, w_2, \dots, w_p]$ and $x = [1, x_1, x_2, \dots, x_p]$
- Assume there are two classes, y = 0 or 1 and let $p_y(x; w)$ be our estimate of $p(y|x)$

$$p_1(x; w) = \frac{1}{1 + \exp(-w^T x)} = \frac{\exp(w^T x)}{1 + \exp(w^T x)}$$

  - $w^T x$ is the dot product
  - for notational simplicity, assume $x_0 = 1$
  - You can define above formula as $p_0(x; w)$, but no difference

- Since $p_1(x; w) + p_0(x; w) = 1$,

$$p_0(x; w) = 1 - \frac{1}{1 + \exp(-w^T x)} = \frac{\exp(-w^T x)}{1 + \exp(-w^T x)} = p_1(x; -w)$$

Logistic Regression

- Since $p_1(x;w) = \frac{1}{1+\exp(-w^T x)}$ and $p_0(x;w) = 1 - \frac{1}{1+\exp(-w^T x)}$
- This is equivalent to the following. (skipped the derivation)

$$\log\left(\frac{p_1(x;w)}{p_0(x;w)}\right) = w^T x$$

- Definition of Linear Classifier (θ: threshold) is

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T \cdot \mathbf{x} > \theta, \\ 0 & \text{otherwise} \end{cases}$$

- That is, logistic regression a linear function(classifier) of x

- Therefore, in logistic regression, we transform the y-axis from the probabilities to log(odds)
- The linear model ($w^T x$) calculates the log(odds) of event Y=1, specifically, the log(odds) that a sample belongs to class 1.

Logistic Regression

- What are odds and log-odds?
  - Odds may be familiar from the field of gambling.
  - Odds are often stated as wins to losses (wins : losses),
    e.g. a one to ten chance or ratio of winning is stated as 1 : 10.

- Given the probability of success (*p*) predicted by the logistic regression model, we can convert it to odds of success as the probability of success divided by the probability of not success:
  - odds of success =$p/(1-p)$
- In Logistic Regression, the logarithm of the odds is calculated, specifically log base-e or the natural logarithm.
- This quantity is referred to as the log-odds and may be referred to as the logit (logistic unit),
  - log-odds = $\log(p/(1-p))$

Recap: Logistic Regression

- For $K = 2$ classes, we had class 1 associated with response $y = 1$ and class 0 be associated with response $y = 0$, and defined

$$p_1(x; w) = p(x; w)$$
$$p_0(x; w) = 1 - p(x; w)$$

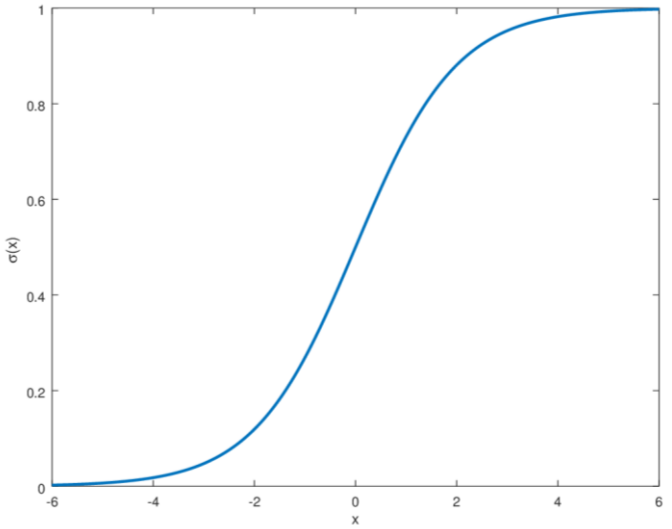
with

$$p(x; w) = \frac{1}{1 + \exp(-w^T x)} = \frac{\exp(w^T x)}{1 + \exp(w^T x)}$$

- Notice that

$$w^T x \gg 0 \implies p(x; w) \gg 0.5 \implies p(x; w) \approx 1$$
$$w^T x \ll 0 \implies p(x; w) \ll 0.5 \implies p(x; w) \approx 0$$

$p(x; w)$



$w^T x$

Recap: Logistic Regression

$$p(x;w) = \frac{1}{1+\exp(-w^T x)} = \frac{\exp(w^T x)}{1+\exp(w^T x)}$$

$$1 - p(x;w) = 1 - \frac{1}{1+\exp(-w^T x)} = \frac{\exp\left(-w^T x\right)}{1+\exp(-w^T x)} = p(x;-w)$$

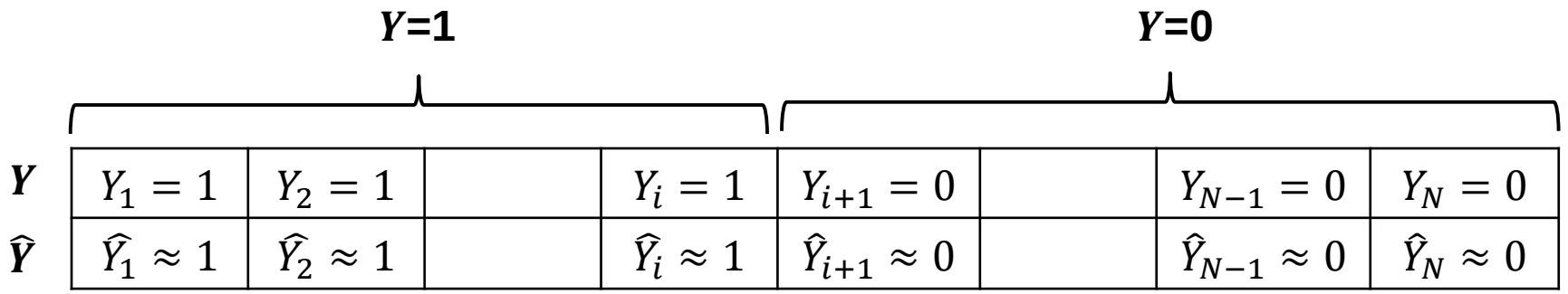
- Also we know that

$$\frac{\partial p(x;w)}{\partial w} = p(x;w)(1-p(x;w))$$

Cost/Loss Function Using MLE

- A case of conditional Maximum Likelihood Estimation(MLE)
- One reasonable approach to training Logistic Regression is to choose parameter values that maximize the conditional data likelihood.
- The conditional data Likelihood is the probability of the observed Y values in the training data, conditioned on their corresponding X values.
- The higher the Likelihood, the higher the probability of observing Y
- We choose the parameters *w* that maximize the log probability of the true Y labels in the training data given the observations *x*

Maximum Likelihood Estimation (MLE)



$$\widehat{Y_i} = p(x_i; w)$$

- Maximum Likelihood:
- We try to find parameter w values that satisfies
  1) If $Y_i = 1$, try to make $\widehat{Y_i}$ as close to as 1
  2) If $Y_i = 0$, try to make $\widehat{Y_i}$ as close to as 0
- **In other words, try to make $\widehat{Y_i}$ as close as to Y_i**

Cost/Loss Function Using Maximum Likelihood

- For a sample labeled as ‘$\mathrm{y} = 1$’, we try to estimate w such that $p(x_i; w)$ is as close to 1 as possible.

$$\max_w p(x_i; w)$$

- For all samples with ‘$\mathrm{y} = 1$’, we maximize

$$\max_w \prod_{s \in \{y_i = 1\}} p(x_i; w) = \max_w p(x_i; w)^{y_i}$$

- For a sample labeled as ‘$\mathrm{y} = 0$’, we try to estimate w such that $p(x_i; w)$ is as close to 0 as possible. In other words, $1 - p(x_i; w)$ should be as close to 1 as possible.

$$\max_w (1 - p(x_i; w))$$

- For all samples with ‘$\mathrm{y} = 0$’, we maximize

$$\max_w \prod_{s \in \{y_i = 0\}} (1 - p(x_i; w)) = \max_w (1 - p(x_i; w))^{(1 - y_i)}$$

Cost/Loss Function Using Maximum Likelihood

- On combining the previous conditions we want to find w parameters such that the product of both of these products is maximum over all elements of the dataset
- Putting these together, we can write the likelihood of parameter vector w

$$L(w) = \prod_{i=1}^{N} p(x_i; w)^{y_i}[1 - p(x_i; w)]^{1-y_i}$$

- We seek w that maximizes $L(w)$,

$$L(w) = \prod_{i=1}^{N} p(x_i; w)^{y_i}[1 - p(x_i; w)]^{1-y_i}$$

which is the same as identifying w that maximizes the log likelihood

$$\ell(w) = \log L(w)$$

$$= \sum_{i=1}^{N} (y_i \log p(x_i; w) + (1 - y_i) \log[1 - p(x_i; w)])$$

Computing the likelihood of the parameters w

- Since $p(x;w) = \frac{1}{1+\exp(-w^T x)} = \frac{\exp(w^T x)}{1+\exp(w^T x)}$, the log likelihood of w is then

$$\ell(w) = \sum_{i=1}^{N} (y_i \log p(x_i;w) + (1-y_i) \log[1 - p(x_i;w)])$$

$$= \sum_{i=1}^{N} \left(y_i x_i^T w - y_i \log[1 + \exp(x_i^T w)] - (1-y_i) \log[1 + \exp(x_i^T w)] \right)$$

$$= \sum_{i=1}^{N} \left(y_i x_i^T w - \log[1 + \exp(x_i^T w)] \right)$$

Gradient of the cost function

$$\ell(w) = \sum_{i=1}^{N} \left(y_i x_i^T w - \log[1 + \exp(x_i^T w)] \right)$$

$$\frac{\partial}{\partial w_j} \ell(w) = \sum_{i=1}^{N} y_i \frac{\partial}{\partial w_j} \left(\sum_{j=1}^{p} x_{ij} w_j \right)$$

$$- \sum_{i=1}^{N} \frac{\partial}{\partial w_j} \log \left[1 + \exp \left(\sum_{j=1}^{p} x_{ij} w_j \right) \right]$$

$$= \sum_{i=1}^{N} \left[y_i x_{ij} - \frac{x_{ij} \exp(x_i^T w)}{1 + \exp(x_i^T w)} \right]$$

$$= \sum_{i=1}^{N} x_{ij} [y_i - p(x_i; w)]$$

Fitting the model parameters w

- Because $\ell(w)$ is concave, $-\ell(w)$ is convex and has a unique global minimum.

- Therefore, we can estimate w that fit training dataset (x_i, y_i), $i = 1,2, \dots, N$, for each $y_i \in \{0,1\}$ by minimizing the cost function

$$J(w) = -\ell(w) = -\sum_{i=1}^{N} (y_i \log p(x_i; w) + (1 - y_i) \log[1 - p(x_i; w)])$$

- Bad News: there is no closed form solution to maximizing l(w) with respect to w

Cross-entropy Loss Function

- The cost function is now as follows:

$$J(w) = -\ell(w) = -\sum_{i=1}^{N} (y_i \log p(x_i; w) + (1 - y_i) \log[1 - p(x_i; w)])$$

- When predicted value is $\hat{y}$ and target value is y, (binary) cross-entropy is defined as

$$L(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log[1 - \hat{y}])$$

- Therefore

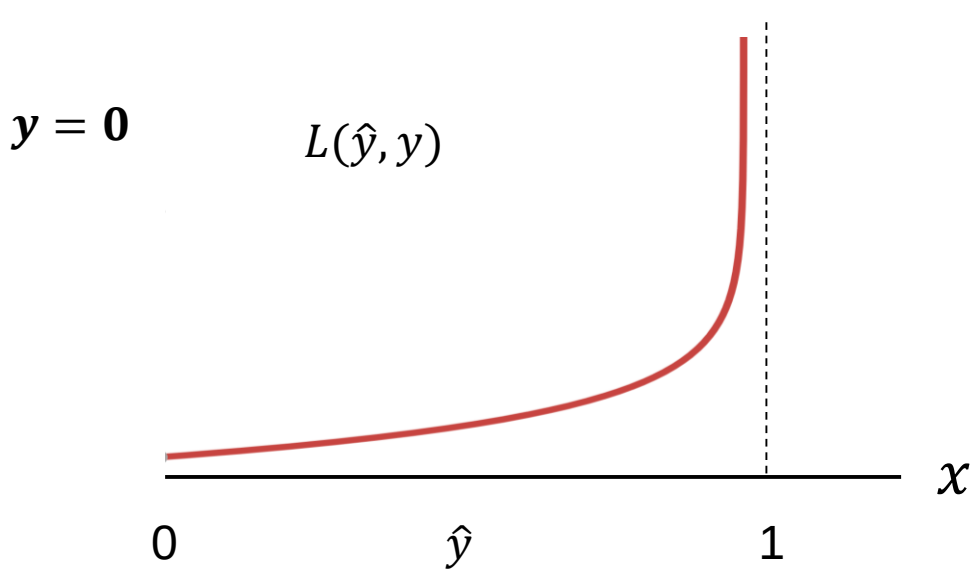
$$J(w) = -\ell(w) = \sum_{i=1}^{N} L(\hat{y}, y)$$

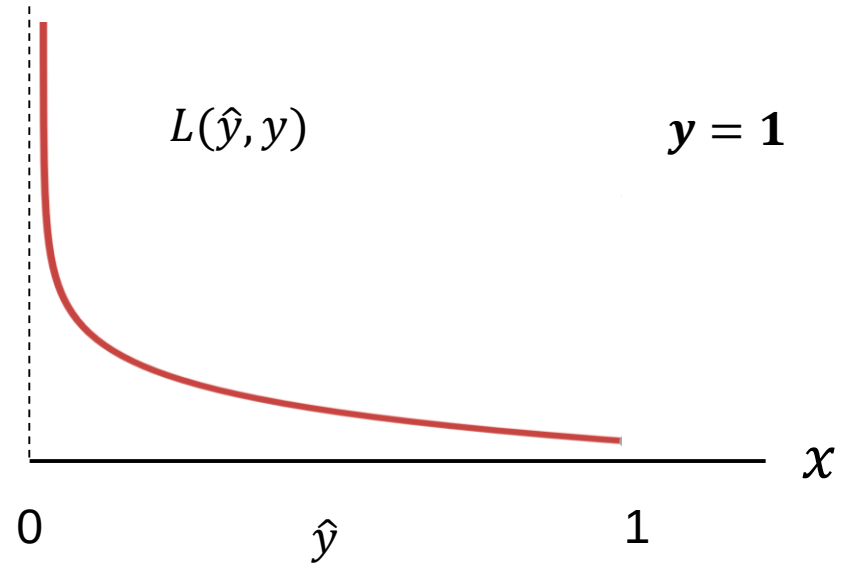
- Logistic Regression uses (binary) **cross-entropy** as the cost/loss function

Cross-entropy Loss

- Suppose predicted value $\hat{y} = p(x; w)$ and target value y is [= either 0 or 1]
- **Goal**: maximize probability of the correct label $p(y|x)$
  - Maximize: $\log p(y|x) = \log\left[\hat{y}^{y}(1-\hat{y})^{1-y}\right] = y \log \hat{y} + (1-y)\log(1-\hat{y})$

- Now flip sign to turn this into a loss: something to minimize
- **Cross-entropy loss** (because is formula for cross-entropy(y, $\hat{y}$))
  - Minimize: $L(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$





Mean Squared Error vs Cross-entropy Loss

- Linear regression uses **mean squared error(MSE)** and Logistic Regression uses **cross-entropy** as its cost function, respectively.
- What if MSE is used for logistic regression?
- Problem 1:
  - The cost function of Logistic Regression with MSE becomes

$$\sum_{i=1}^{N}(y_i - \log p(x_i; w))^2$$

  - It is a **non-convex** function of parameters. Therefore, local optimum
- Problem 2:
  - MSE used in logistic regression does not penalize the model in a great way
  - Suppose your LR model predicts a probability of class 1 as 0.1 (target value=1)
  - Ideally, for a good cost function, the penalization/error value should be high.
  - Let's compare the MSE and LogLoss

MSELOSS	LogLoss
=(Y_Actual-Y_Pred)^2	=-(Y_Actual * log(Y_Actual))
=(1-0.1)^2	=-(1 * log(0.1))
=0.81	= 2.302

Finding w with gradient descent

- We can employ (batch, stochastic, mini-batch) gradient descent to fit the model parameters w to training data (no closed-form solution).

Fix learning rate α

1. Randomly initialize $w = [w_0, w_1, \dots, w_p]$.

2. Update jth parameter $j \in \{0,1, \dots, p\}$ as

$$w_j \coloneqq w_j - \alpha \frac{\partial}{\partial w_j} J(w) = w_j + \alpha \frac{\partial}{\partial w_j} \ell(w)$$

3. Repeat step 2 until convergence.

- To implement gradient descent for logistic regression with $K = 2$ classes, we need to compute the partial derivative of the log likelihood function $\ell(w)$ with respect to each parameter w_j. (i.e., $\frac{\partial}{\partial w_j} \ell(w)$)

Finding w with gradient descent

Fix learning rate α

1. Randomly initialize $w = [w_0, w_1, \dots, w_p]$.

2. Update jth parameter $j \in \{0,1, \dots, p\}$ as

$$w_j := \begin{cases} w_0 + \alpha \sum_{i=1}^{N} [y_i - p(x_i; w)] & \text{if } j = 0 \\ w_j + \alpha \sum_{i=1}^{N} x_{ij} [y_i - p(x_i; w)] & \text{if } j = 1,2, \dots, p \end{cases}$$

3. Repeat step 2 until convergence.

- Above update rule can be simplified as follows by assuming $x_{i0} = 1$

$$w_j := w_j + \alpha \sum_{i=1}^{N} x_{ij} [y_i - p(x_i; w)]$$

Regularization in Logistic Regression

- So far we have maximized training accuracy.

- Overfitting the training data is a problem that can arise in Logistic Regression, especially when data has very high dimensions and is sparse.
  - Prefer simple, compact model

- One approach to reducing overfitting is regularization, in which we create a modified "penalized log likelihood function," which penalizes large values of w.

$$J(w) = -\sum_{i=1}^{N}(y_i \log p(x_i; w) + (1 - y_i) \log[1 - p(x_i; w)]) + \frac{\lambda}{2}\|w\|^2$$

which adds a penalty term (L2 penalty term) proportional to the squared magnitude of w. Here λ is a constant that determines the strength of this penalty term.

Regularization in Logistic Regression

- The derivative of this penalized $J(w)$ function is similar to our earlier derivative, with one additional penalty term

$$\frac{\partial}{\partial w_j}\ell(w) = \sum_{i=1}^{N} x_{ij}[y_i - p(x_i; w)] - \lambda w_i$$

  which gives us the modified gradient descent rule

$$w_j = w_j + \alpha \sum_{i=1}^{N} x_{ij}[y_i - p(x_i; w)] - \alpha\lambda w_i$$

Logistic Regression with Many Discrete Values

- So far we considered using Logistic Regression to learn P(Y|X) only for the case where Y is a boolean variable.

- More generally, if Y can take on any of the discrete values $\{y_1, y_2, \ldots y_{k-1}\}$, then the form of $p_k(x; w)$ for $Y = y_1, y_2, \ldots y_{k-1}$ is:

$$p_k(x; w) = \frac{\exp(w_{k0} + \sum_{i=1}^{n} w_{ki} x_i)}{1 + \sum_{j=1}^{k-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji} x_i)} = \frac{\exp(w_k^T x)}{1 + \sum_{j=1}^{k-1} \exp(w_j^T x)}$$

- When $Y = y_k$, it is

$$p_k(x; w) = \frac{1}{1 + \sum_{j=1}^{k-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji} x_i)}$$

- When Y takes on K possible values, we construct K-1 different linear expressions to capture the distributions for the different values of Y.

- The distribution for the final, Kth, value of Y is simply one minus the probabilities of the first K-1 values.

Logistic Regression with Many Discrete Values

- For notational simplicity, suppose class values are 1 and 2 (instead of 0 and 1). Binary Logistic Regresson maximizes

$$\prod_{i=1}^{N} p(x_i; w)^{y_i}[1 - p(x_i; w)]^{1-y_i} = \prod_{i=1}^{N} p_1(x_i; w)^{y_i} p_2(x_i; w)^{1-y_i}$$

- Now we have a general case of K classes. Let y_k, $k = 1, 2, \ldots, K$ be an indicator variable determining whether an observation derives from category k and let y_{ik} be an indicator variable with $y_{ik} = 1$ if $y_i = k$ and 0 otherwise.

- Now we maximize

$$L(w) = \prod_{i=1}^{N} \prod_{k=1}^{K} p_k(x_i; w)^{y_{ik}}$$

- We seek w that maximizes the log likelihood

$$\ell(w) = \log L(w) = \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \log p_k(x_i; w)$$

Logistic Regression with Many Discrete Values

- Because $\ell(w)$ is concave, $-\ell(w)$ is convex and has a unique global minimum.

- Therefore, we can estimate θ that fit training dataset (x_i, y_i), $i = 1,2, \dots, N$, for each $y_i \in \{1,2, \dots, K\}$ by minimizing the cost function

$$J(w) = -\ell(w) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log p_k(x_i; w)$$

- Recall that $w_k \in \mathbb{R}^{p+1}$ with $w_k = [w_{k0}, w_{k1}, \dots, w_{kp}]$ and that w is defined as the collection $w = \{w_1, w_2, \dots, w_K\}$

- Here w_{jk} denotes the weight associated with the jth class $Y = y_k$ and with input x_i

Finding w with gradient descent

Fix learning rate α

1. Randomly initialize $w = \{w_1, w_2, \dots, w_K\}$.

2. For class $k \in \{1,2, \dots, K\}$, update jth parameter $j \in \{0,1, \dots, p\}$ as

$$w_{kj} \coloneqq w_{kj} - \alpha \frac{\partial}{\partial w_{kj}} J(w)$$

$$= w_{kj} + \alpha \frac{\partial}{\partial w_{kj}} \ell(w)$$

3. Update θ as $w = \{w_1, w_2, \dots, w_K\}$.

4. Repeat steps 2 and 3 until convergence.

Computing the gradient of the cost function

Skipping the derivation, for $j \in \{1,2,\dots,p\}$, we have

$$\frac{\partial}{\partial w_{kj}} \ell(w) = \sum_{i=1}^{N} x_{ij}[y_{ik} - p_k(x_i; w)]$$

Finding w with batch gradient descent

Fix learning rate α 1. Randomly initialize $w = \{w_1, w_2, \dots, w_K\}$.

2. For class $k \in \{1,2, \dots, K\}$, update jth parameter $j \in \{0,1, \dots, p\}$ as

(assuming $x_{i0} = 1$)

$$w_{kj} := w_{kj} + \alpha \sum_{i=1}^{N} x_{ij}[y_{ik} - p_k(x_i; w)]$$

3. Update w as $w = \{w_1, w_2, \dots, w_K\}$.

4. Repeat steps 2 and 3 until convergence.

Logistic Regression in sklearn

```
import numpy as np
from sklearn.linear_model import LogisticRegression

x = np.arange(10).reshape(-1, 1)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

model = LogisticRegression()
model.fit(x, y)

print(f"class: {model.classes_}") #  class labels
print(f"intercept: {model.intercept_}") # bias
print(f"slope: {model.coef_}") # Coefficient of the features

print(model.predict_proba(x))
print(model.predict(x))
print(f"score: {model.score(x, y)}")
```

```
class: [0 1]
intercept: [-4.12617727]
slope: [[1.18109091]]

[[0.98 0.02]
 [0.95 0.05]
 [0.85 0.15]
 [0.64 0.36]
 [0.35 0.65]
 [0.14  0.86]
 [0.04 0.96]
 [0.01 0.99]
 [0.01 0.99]
 [0.01 0.99]]
[0 0 0 0 1 1 1 1 1 1]
score: 1.0
```

Logistic Regression in sklearn

model = LogisticRegression(penalty='l2', C=1.0)

- **penalty**: Specify the norm of the penalty
  - None: no penalty is added;
  - 'l2': add a L2 penalty term. (default)
  - 'l1': add a L1 penalty term;
  - 'elasticnet': both L1 and L2 penalty terms are added.

- **C**: Inverse of regularization strength;
  - It's $1/\lambda$, where λ is the regularization strength.
  - must be a positive float.
  - Smaller values specify stronger regularization. (default=1.0)