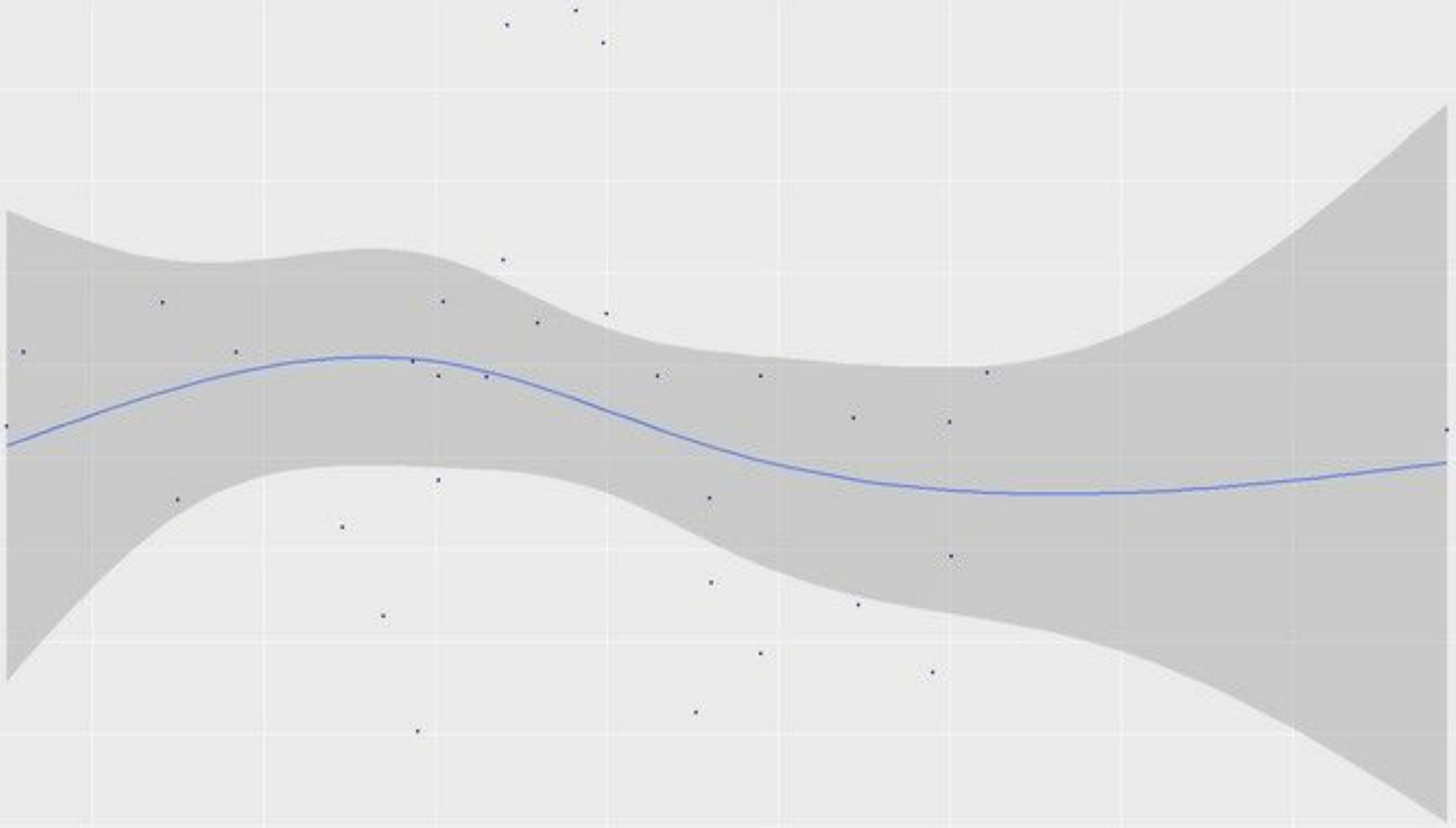
Linear Models

Ayush Thada (16BCE1333)



Content

- Linear Regression
- Perceptron
- Logistic Regression
- Additional Resources

Linear Regression

- **Definition**: Linear regression is a linear approach to modeling the relationship between a scalar response (or **dependent** variable) and one or more explanatory variables (or *independent* variables). The case of one explanatory variable is called **simple linear regression**. For more than one explanatory variable, the process is called **multiple linear regression**.

- **Assumption for data**:

$$Y = W^T . X + \in \quad \text{(Vector Representation)}$$

$$y = w_0 + w_1x_1 + w_2x_2 + \dots + w_{k-1}x_{k-1} + \in$$

where

w_i ← weights ; $i = 0, 1, 2 \dots , k-1$

$\in$ ← random noise; $\in \sim N(0, \sigma^2)$

So data distribution can be represented as :

$$y \sim N(w_0 + w_1x_1 + w_1x_2 + \dots + w_{k-1}x_k , \sigma^2)$$

Here we also assume that each data point that we have is independently and identically sampled from above distribution.

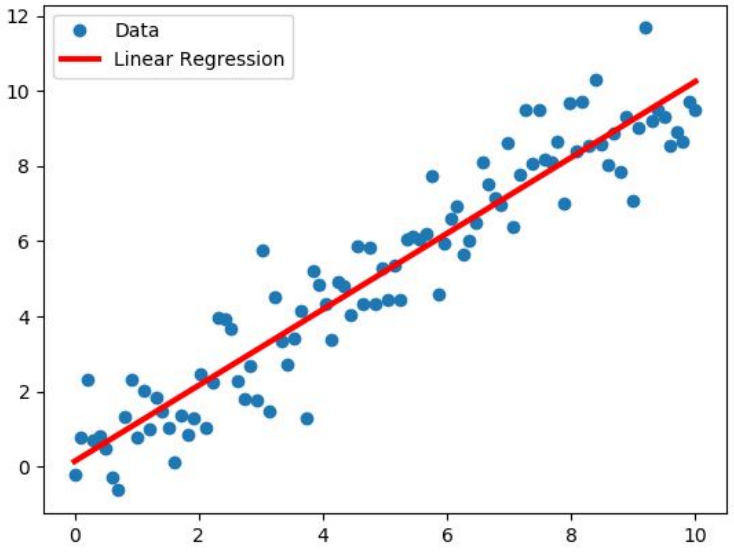
- **Model Prediction**:

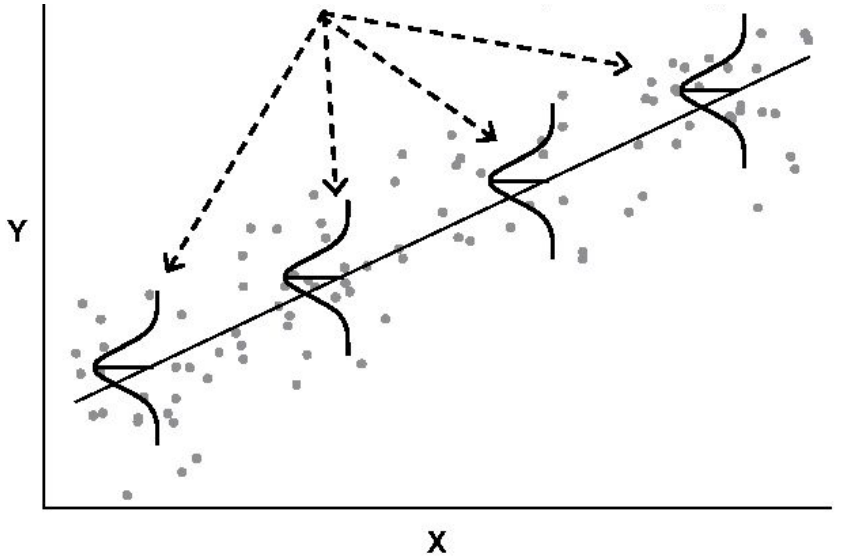
$$\hat{Y} = \hat{W}^T . X \quad \text{(Vector Representation)}$$

$$\hat{y} = \hat{w}_0 + \hat{w}_1 x_1 + \hat{w}_2 x_2 + \dots + \hat{w}_{k-1} x_{k-1}$$

where

$\hat{w}_i$ ← calculated weights ; $i = 0, 1, 2 \dots , k-1$





- **Common terminology:**
  - *Residuals*: Error in the prediction is called residuals.

$$e_i = y_i - \hat{y}_i$$

$$E = Y - \hat{Y}$$

  - *SSE*: Sum of squared errors or residuals.

$$SSE = \Sigma (y_i - \hat{y}_i)^2$$

$$SSE = \Sigma (y_i - \hat{w}_0 + \hat{w}_1 x_{1i} + \hat{w}_2 x_{2i} + \ldots + \hat{w}_{k-1} x_{(k-1)i})^2$$

- **Learning:**

To learn the parameters of the model, we try to **minimize the SSE** with respect to parameters of the model.

First let's discuss it for simple linear regression where there is only two parameters **bias/intercept and slope**. Then we can generalize it for multi linear regression.

1. Simple Linear Regression:

$$S = SS_{Res} = \Sigma e^2 = \Sigma (y - \hat{y})^2 = \Sigma (y - w_0 - w_1 x)^2$$

$$\partial S/\partial \hat{w}_0 = -2 \cdot \Sigma(y_i - \hat{w}_0 - \hat{w}_1 x_i) \text{ (I normal equation)}$$

$$\partial S/\partial \hat{w}_1 = -2 \cdot \Sigma x_i(y_i - \hat{w}_0 - \hat{w}_1 x_i) \text{ (II normal equation)}$$

- Take first normal equation:

$$\partial S/\partial \hat{w}_0 = 0$$

$$-2 \cdot \Sigma(y_i - \hat{w}_0 - \hat{w}_1 x_i) = 0$$

$$\Sigma y_i - n.\hat{w}_0 - \hat{w}_1 \Sigma x_i = 0$$

$$\mathbf{\hat{w}_0 = \bar{y} - \hat{w}_1 \bar{x}}$$

- Take second normal equation:

$$\partial S/\partial \hat{w}_1 = 0$$

$$-2 \cdot \Sigma x_i(y_i - \hat{w}_0 - \hat{w}_1 x_i) = 0$$

$$\Sigma x_i(y_i - \bar{y} + \hat{w}_1 \bar{x} - \hat{w}_1 x_i) = 0$$

$$\Sigma x_i(y_i - \bar{y}) = \hat{w}_1 \Sigma x_i(x_i - \bar{x})$$

$$\hat{w}_1 = \Sigma x_i(y_i - \bar{y}) \,/\, \Sigma x_i(x_i - \bar{x})$$

$$\hat{w}_1 = \Sigma x_i(y_i - \bar{y}) \,/\, \Sigma(x_i - \bar{x})^2$$

$$\hat{w}_1 = \Sigma y_i(x_i - \bar{x}) \,/\, \Sigma(x_i - \bar{x})^2$$

$$\hat{w}_1 = \Sigma(x_i - \bar{x})(y_i - \bar{y}) \,/\, \Sigma(x_i - \bar{x})^2 = S_{xy} / S_{xx}$$

where

$$S_{xy} = \Sigma(x_i - \bar{x})(y_i - \bar{y})$$

$$S_{xx} = \Sigma(x_i - \bar{x})^2$$

Last three steps can be proved easily from the first equation.

2. Multiple Linear Regression:

In this case, we're going to use vector notation for the derivation of parameters of regression model.

$$SS_{Res} = \Sigma e_i^2$$

$$SS_{Res} = e^T e$$

$$SS_{Res} = (Y - \hat{Y})^T (Y - \hat{Y})$$

$$SS_{Res} = (Y - X\hat{W})^T (Y - X\hat{W})$$

$$SS_{Res} = Y^T Y - 2\hat{W}^T X^T Y - \hat{W}^T X^T X B = S$$

$$\partial S/\partial \hat{W} = -2X^T Y + 2X^T X\hat{W} \text{ (Normal Equation)}$$

Now we have to set the $\partial S/\partial \hat{W} = 0$.

$$\partial S/\partial \hat{W} = 0$$

$$-2X^T Y + 2X^T X\hat{W} = 0$$

$$\hat{W} = (X^T X)^{-1} X^T Y$$

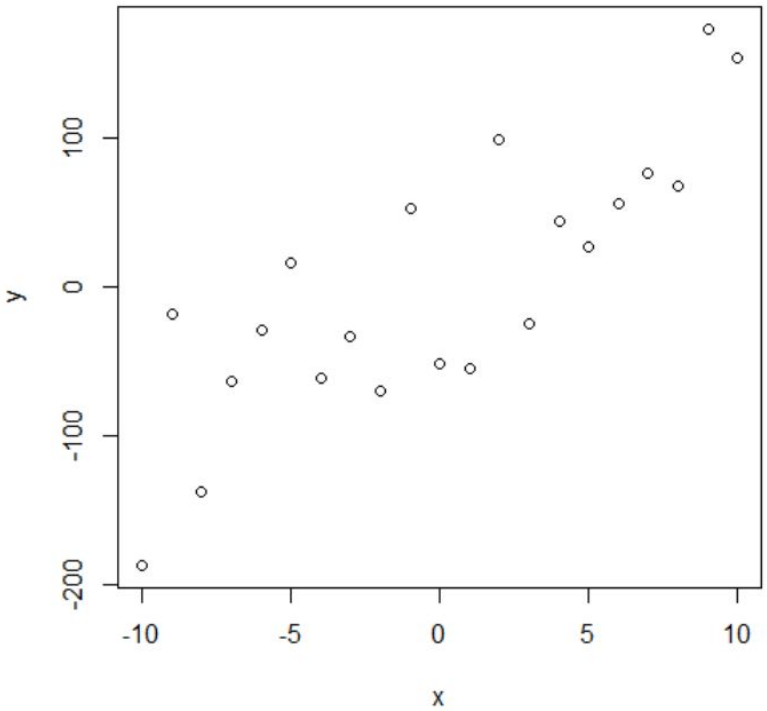
The parameters that we have received in the case of both regression are called **least square estimators**.

- **<u>Model Verification:</u>**

  For the verification we can do variety of tests:

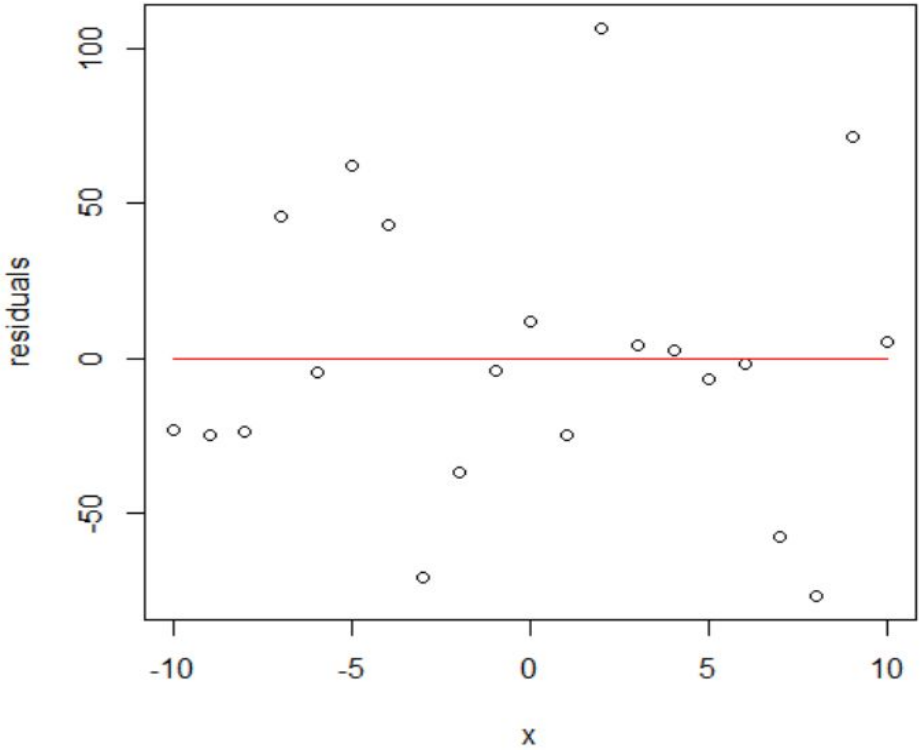
  - P- test
  - T- test
  - F-test
  - Anova
  - Inference using R squared error

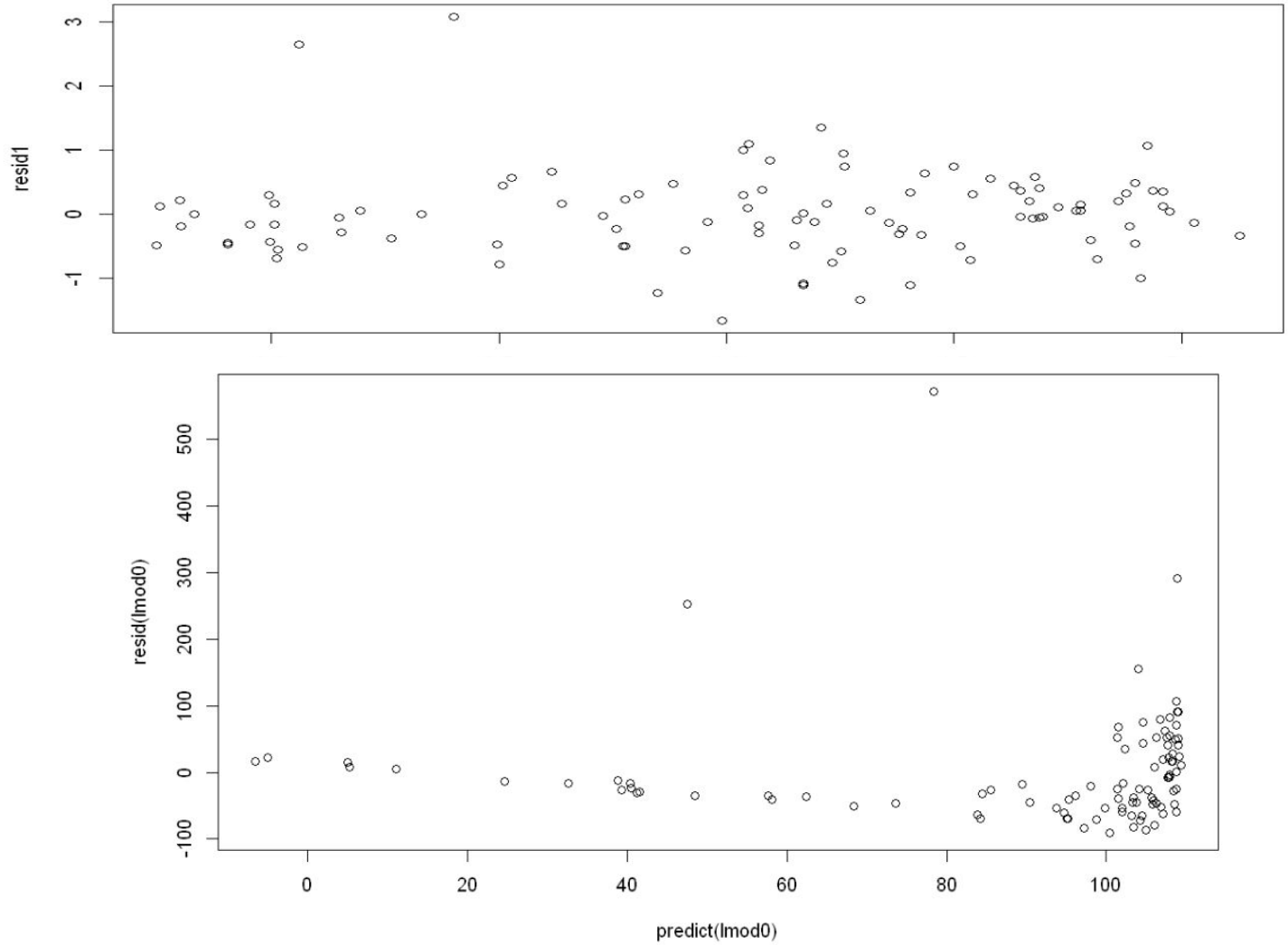
```
> x <- seq(-10, 10)
> y <- rnorm(n = length(x), mean = 15*x + 7, sd = 49)
> model <- lm(y ~ x)
> summary(model)

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-76.595 -24.587  -4.011  12.214 106.684

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   25.619     10.304   2.486   0.0224 *
x             14.723      1.702   8.652 5.13e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 47.22 on 19 degrees of freedom
Multiple R-squared:  0.7976,    Adjusted R-squared:  0.7869
F-statistic: 74.86 on 1 and 19 DF,  p-value: 5.133e-08
```



But the performance of the model can be roughly evaluated with the help of visualization. To o the task we can plot the residuals and check for patterns in the plot. There should be no trend in the residuals.

```
> y_hat <- predict(model)
> residuals <- y - y_hat
> plot(x, residuals)
> lines(x, rep(0,length(x)), col="red")
```




resid1
3
2
1
0
-1
resid(lmod0)
500
400
300
200
100
0
-100
0
20
40
60
80
100
predict(lmod0)

- **Extension and Other Variants:**

    1. Heteroscedastic models
    2. Generalized Linear Models
        a. Poisson Regression
        b. Logistic Regression
    3. Hierarchical linear models
    4. Least Angle Regression
    5. Bayesian Linear Regression
    6. Principal component Regression

Perceptron

- **Definition:** The perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class.

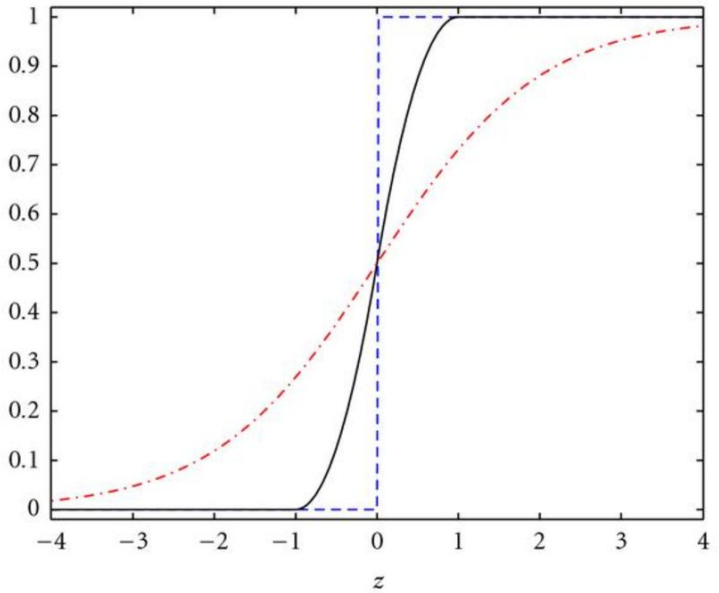
$$F(x) = \mathbb{1}[W^T . X] \quad \text{(Vector Representation)}$$

$$F(x) = \mathbb{1}[w_0 + w_1x_1 + w_2x_2 + \ldots + w_{k-1}x_{k-1}]$$

where

$\mathbb{1}[x]$ represent indicator/Heaviside function.

$$\mathbb{1}[x] = \frac{d}{dx}\max\{x, 0\} \quad \text{for } x \neq 0$$



Indicator $I(z > 0)$
$h(z)$
Sigmoid $g(z)$

- **Learning Algorithm [Gradient Descent]:**

Step 1: Initialize weights to small numbers such that | w | < 1.

Step 2: Input a feature vector and do inner product with corresponding weights and add the products and bias.

$$\textbf{output} = W^T X + b$$

Step 3: Calculate the heaviside function of previous output ie. max(output, 0).

$$\textbf{prediction} = \textbf{activation} = \mathbb{1}(\textbf{output})$$

Step 4: Now we have to update the weights by this rule

$$W := W - \text{learning\_rate} * (\text{expected\_label} - \text{predicted\_label}) * X$$

Step 5: Repeat from Step 2 to Step 5 until convergence.

Step 6: Return weights.

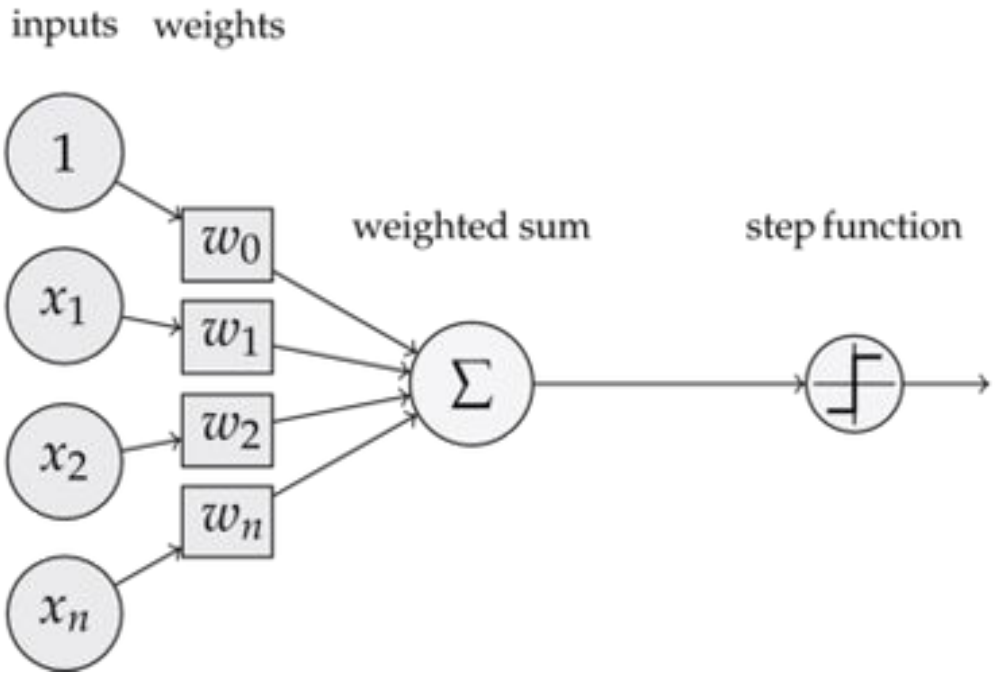


Figure: **Visualization of Perceptron Algorithm**

- **<u>Gradient Descent Explanation:</u>**

It is a numerical method to find the critical point of a function. It is used when wehn solution predicted through the analytical method is difficult to calculate or it is difficult to derive an analytical solution.

Assume we have given a dataset from which we have derived features X and we have associated label as Y. For simplicity we can assume case of linear regression

Cost function is

$$Y = wx + b$$

$$C(Y, \hat{Y}) = (1/N) \cdot \Sigma(y_i - wx - b_i)^2$$

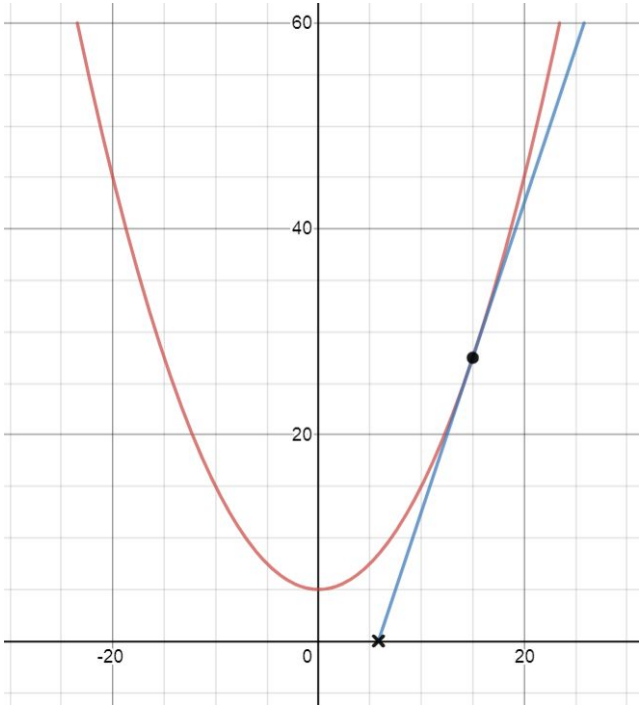
Let's calculate the gradients with respect to parameters **w** and **b.**

$$\partial C/\partial w = -(2/N) \cdot \Sigma x_i (y_i - \hat{w}_0 - \hat{w}_1 x_i)$$

$$\partial C/\partial w = -(2/N) \cdot \Sigma x_i (y_i - \hat{y})$$

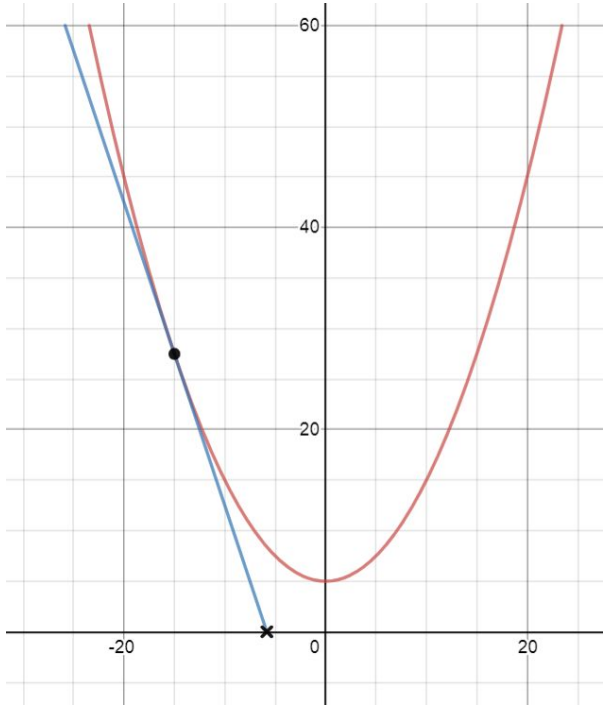
$$\partial C/\partial b = -(2/N) \cdot \Sigma(y_i - \hat{w}_0 - \hat{w}_1 x_i)$$

$$\partial C/\partial b = -(2/N) \cdot \Sigma(y_i - \hat{y}_i)$$



In this case to move toward 0 (ie. critical point),
We update ut current position in this way:

x := x - c.(slope of line)



In this case to move toward 0 (ie. critical point),
We update ut current position in this way:

x := x - c.(slope of line)

There are some variants of gradient descent algorithm:

1. ***Vanilla Gradient Descent***: It is the standard version of gradient descent. In this version we use all the data that we have

   - Cost function is

   $$\hat{Y} = W^T X$$

   $$C(Y, \hat{Y}) = (1/N).\ (Y - \hat{Y})^T (Y - \hat{Y})$$

   - Gradients with respect to all weights

   $$\partial C/\partial W = (2/N)(Y - \hat{Y})$$
   $$\partial C/\partial W = (2/N)(Y - W^T X)$$

   - Update Rule

   $$W := W - learning\_rate\ .\ (Y - W^T X)$$

2. ***Stochastic Gradient Descent***: In this variant of algorithm we use one example at a time to update the weights.

3. ***Mini Batch Gradient Descent***: In this variant we use a subset of dataset to update the weights of the model. The other two variants are the special cases of this version.

Logistic Regression

- **Definition:** Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes)

- **Explanation as a case of Linear Regression**:

  In Logistic regression :

  $$y \sim \text{Bernoulli}(\Phi)$$

  Here $\Phi \in [0,1]$

  Also, the expected value of y is $\mathbb{E}[y] = \Phi$

  Here we also assume that each data point that we have is independently and identically sampled from above distribution. In this case every outcome is the probability that an example belong to particular class.One option is to model Φ directly to the linear form. But there is a problem, Φ represents probability, it needs to lie between 0 and 1. TO ensure this we have to top keep odd restriction on the weights of the model. But rather than this we can use link function which relates linear form of the restriction parameter in our case phi without any restriction on weights.

There are some suggestion for link function

1. Odd ratio:
   In case of bernoulli distribution it is equal to (**Φ / 1 - Φ**).
   But it doesn't take all real value. To use this as expected value of $\hat{\mathbf{Y}}$ we have to still put restriction on weights of model. It relaxed some restrictions but not all.

2. Logit:
   In case of bernoulli distribution it is equal to **log(Φ / 1 - Φ).**
   It can take all real values. If we use this we don't have to put restrictions on our weights.

$$\log(\Phi / 1 - \Phi) = \hat{Y} = W^T X$$

$$(\Phi / 1 - \Phi) = \exp(W^T X)$$

$$\Phi = \exp(W^T X) / (1 + \exp(W^T X))$$

$$\Phi = 1 / (1 + \exp(-W^T X))$$

$$\Phi = \text{sigmoid}(W^T X)$$

- **Cost Function:**

  Here the result of the model is probability, so we'll use cross entropy cost here rather than mean squared error. Before going for cross entropy we have to understand the meaning of information and entropy. This term is borrowed form Information theory.

  - ***Information***:
    It represents the number of bits needed to identify an event.

$$I(X) = -\log_2(P(X))$$

  - ***Entropy***:
    Expected no of bits required to encode class of randomly drawn sample from a probability distribution.

$$S(P) = -\Sigma P(x).\log_2(P(x))$$

  - ***Cross Entropy:***
    Expected number of bits needed to identify an event drawn from the set if a coding scheme used for the set is optimized for an estimated probability distribution q, rather than the true distribution p

$$H(P, Q) = -\Sigma P(x).\log_2(Q(x))$$

We take cost as Kullback–Leibler Divergence:

$$KL(p \,||\, q) = Cross\_Entropy(p, q) - Entropy(p)$$
$$KL(p \,||\, q) = -\Sigma P(x).\log_2(Q(x)) + \Sigma P(x).\log_2(P(x))$$

Let's convert log base 2 to natural log and substitute P and Q distribution with our true label/distribution and predicted labels/distribution respectively.

$$C(y, \hat{y}) = (-\Sigma y_i.\log(\hat{y}_i) + \Sigma y_i.\log(y_i)) \,.\, k$$

The second term is constant and it will not affect our gradient, so we can remove it from our cost term and also we can remove constant term k which is used to convert base 2 to base e.

$$C`(y, \hat{y}) = -\Sigma y_i.\log(\hat{y}_i)$$

Since in case of logistic regression we have only two two classes we can write our expression in uncondensed form

$$C`(y, \hat{y}) = -\,(y_i.\log(\hat{y}_i) + (1 - y_i).\log(1 - \hat{y}_i))$$

- **Learning Algorithm [Gradient Descent]:**

Step 1: Initialize weights to small numbers such that | w | < 1.

Step 2: Input a feature vector and do inner product with corresponding weights and add the products and bias.

$$\text{output} = W^T X + b$$

Step 3: Calculate the heaviside function of previous output ie. max(output, 0).

$$\text{prediction} = \text{activation} = \text{sigmoid}(\text{output})$$

Step 4: Now we have to update the weights by this rule

$$W := W - \text{learning\_rate} * \text{gradient}(C(y, \hat{y}))$$

Step 5: Repeat from Step 2 to Step 5 until convergence.

Step 6: Return weights.

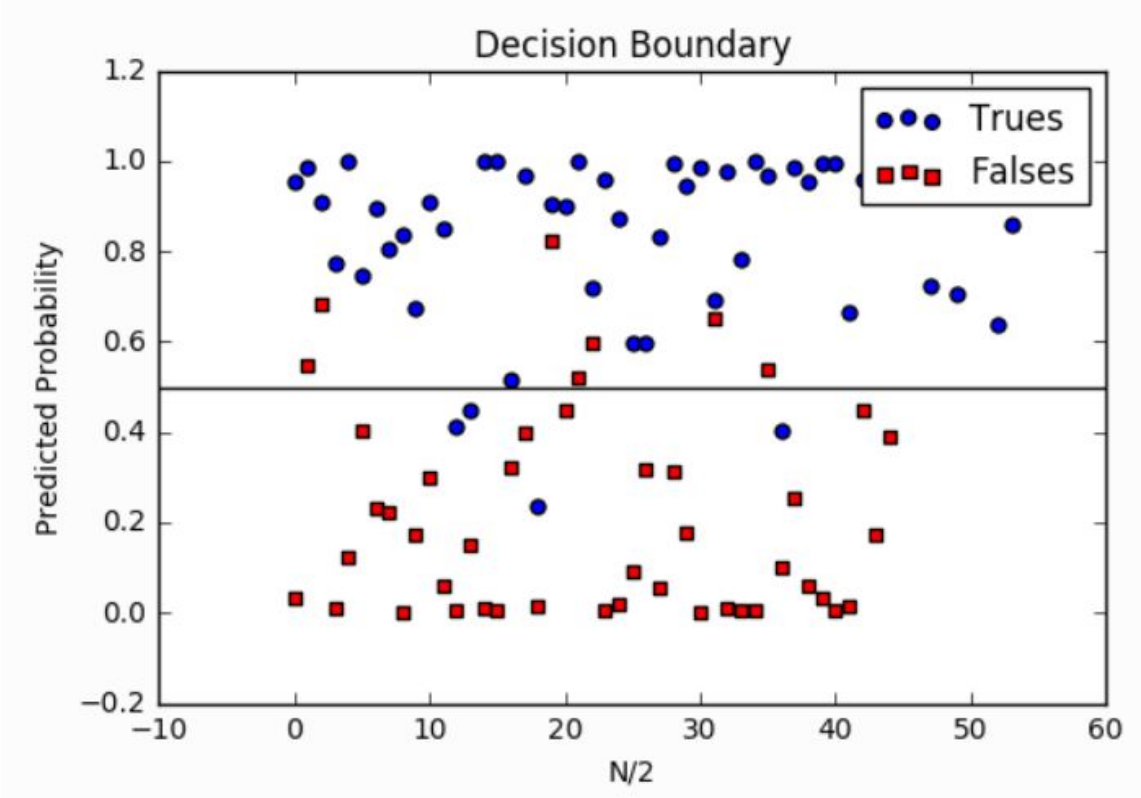


Figure: ***Decision boundary separating linearly separable classes.***

Additional Resources

- **Linear Regression:**
  - ***Chapter 1****: Applied Regression Analysis, Draper and Smith*
  - https://statistics.laerd.com/spss-tutorials/multiple-regression-using-spss-statistics.php
  - http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/BS704_Multivariable/BS704_Multivariable7.html
  - https://ismayc.github.io/teaching/sample_problems/mlr.html

- **Perceptron:**
  - https://www.pearsonhighered.com/assets/samplechapter/0/1/3/1/0131471392.pdf
  - *Original Paper:* https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf
  - *Proff. Ali Ghodsi:* https://www.youtube.com/watch?v=-8Q0h6_r02Y
  - *Michael Nielsen*: http://neuralnetworksanddeeplearning.com/chap1.html
  - ***Chapter 4.4*** *Machine Learning, Tom Mitchell*

- **Logistic Regression:**
  - http://www.holehouse.org/mlclass/06_Logistic_Regression.html
  - https://github.com/perborgen/LogisticRegression/blob/master/logistic.py
  - https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html
  - *Proff. Ali Ghodsi*: https://www.youtube.com/watch?v=w3xbI-OseCI
  - *Proff. Yaser Abu-Mostafa*: https://www.youtube.com/watch?v=qSTHZvN8hzs
  - ***Chapter 8****, Machine Learning A Probabilistic Perspective, Kevin P. Murphy*

Questions