Machine Learning Reference for R

Contents

Data Preparation

Normalization

Features sometimes need to be scaled so they fit into a standard range. This involves transforming variables into a narrower or wider range than they are found in the observed data.

One method for scaling features is **min-max normalization**, which uses the minimum and maximum values within the feature to produce a value between 0 and 1:

$$X_{new} = \frac{X - min(X)}{max(X) - min(X)}$$

```
x <- seq(10, 30, by = 1)
x.new <- (x - min(x)) / (max(x) - min(x))
```

This method is useful when the values different features are required to be within the same range, for example with K-Nearest Neighbors classifiers and K-Means clustering.

Another method for scaling features is **z-score normalization**, which standardizes the feature to have the features of the normal distribution (with a mean of 0 and standard deviation of 1):

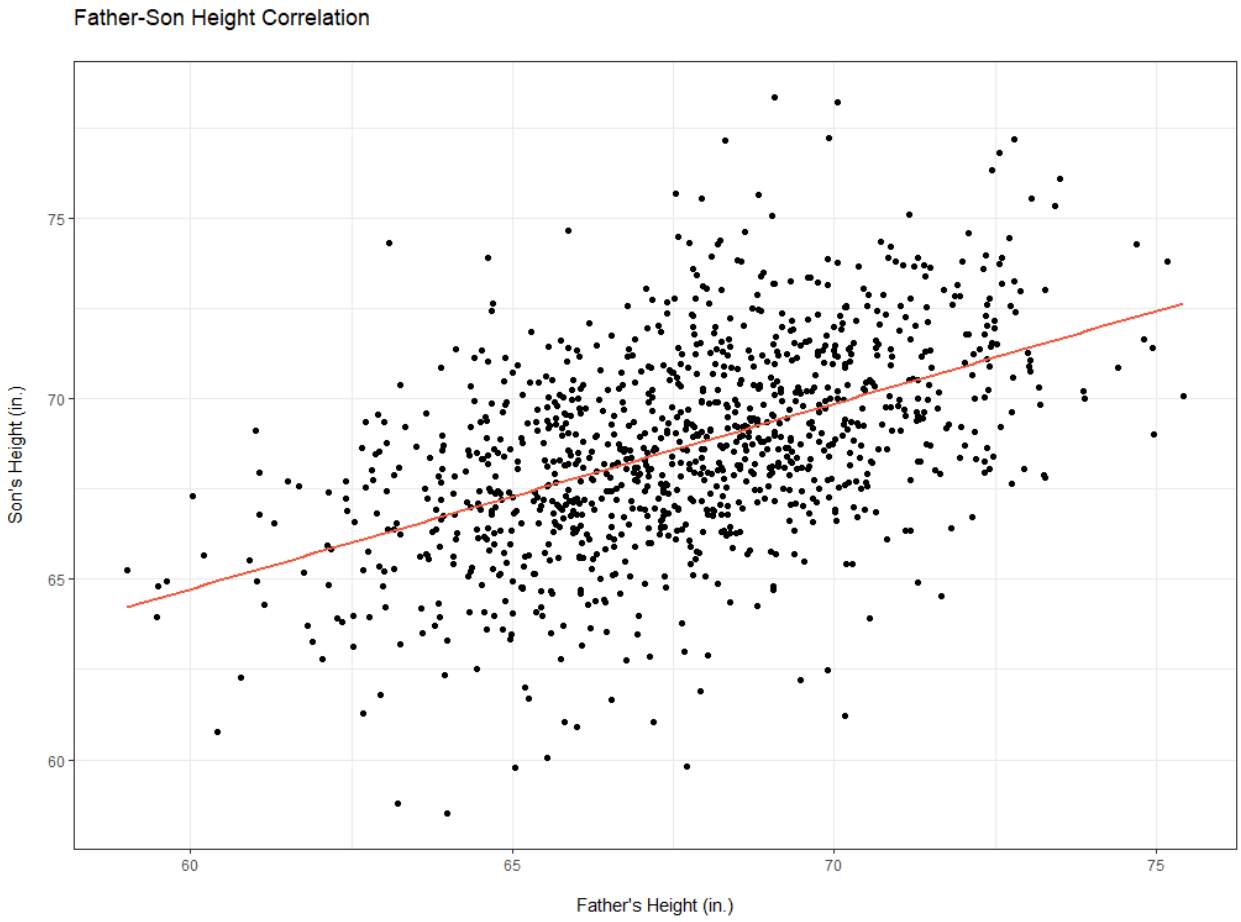
$$X_{new} = \frac{X - \mu}{\sigma} = \frac{X - mean(X)}{StdDev(X)}$$

```
x <- seq(5, 45, by = 1)
x.new <- (x - mean(x)) / sd(x)
```

Algorithms

Regression Algorithms

Simple Linear Regression



Simple linear regression is a statistical method for evaluating the relationship between two continuous variables. One variable, denoted x, is the **independent variable** (also called **predictor variable**) that is used to predict the **dependent variable** (also called **response variable**), denoted y.

In linear regression, we use methods such as **ordinary least squares** to approximate the equation of a line describing the relationship between the predictor and response variables. We find a **correlation coefficient**:

$$r = \frac{1}{n-1} \sum \left(\frac{x - \bar{x}}{s_x} \right) \left(\frac{y - \bar{y}}{s_y} \right)$$

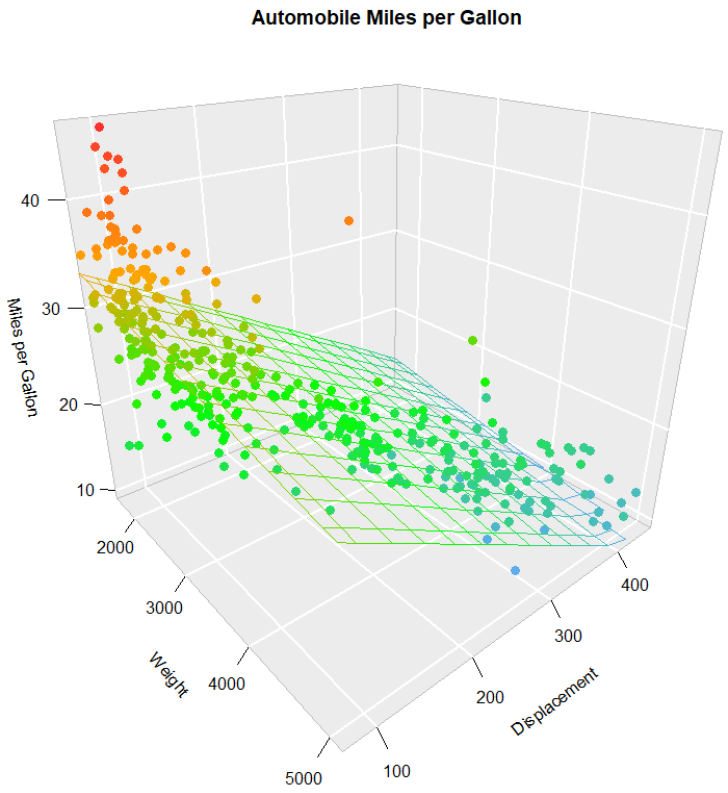
In simple linear regression, we estimate two coefficients, β_0 (the y-intercept) and β_1 (the regression slope). Because this is an estimate, there is a residual error, ϵ. These are combined to form a **line of best fit**:

$$\hat{y_i} = \beta_0 + \beta_1 x_i + \epsilon$$

Functions related to running simple linear regression models in R are:

Function	Package	Description	Example
`cor()`	stats	Calculate correlation coefficient.	`cor(x, y)`
`lm()`	stats	Train a linear regression model.	`lm(sheight ~ fheight, data = father.son)`
`predict()`	stats	Predict values using the trained model.	`m <- lm(y ~ x)` `predict(m, test.data)`
`summary()`	base	Summarize a linear model.	`m <- lm(y ~ x)` `summary(m)`
`confint()`	stats	Compute confidence interval for model parameters.	`m <- lm(y ~ x)` `confint(m)`

Multiple Linear Regression



Multiple linear regression is a statistical method for evaluating the relationship between a response (dependent) variable and two or more predictor (independent) variables. It generalizes simple linear regression by allowing for the use of multiple predictor variables instead of just one.

In multiple linear regression, we estimate a β_0 coefficient (the y-intercept), as well as a β_i coefficient for each of n predictor variables. This produces a **line of best fit**:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \epsilon$$

$$= \beta_0 + \sum_{i=1}^{n} \beta_i x_i + \epsilon$$

Functions related to running multiple linear regression models in R are:

Function	Package	Description	Example
`cor()`	stats	Calculate correlation coefficient.	`cor(x, y)`
`lm()`	stats	Train a linear regression model.	`lm(z ~ x + y, data = my.data)`
`predict()`	stats	Predict values using the trained model.	`m <- lm(z ~ x + y)` `predict(m, test.data)`
`summary()`	base	Summarize a linear model.	`m <- lm(y ~ x)` `summary(m)`
`confint()`	stats	Compute confidence interval for model parameters.	`m <- lm(z ~ x + y)` `confint(m)`

Classification Algorithms

Naive Bayes

The **Naive Bayes classifier** is a probabilistic machine learning algorithm that predicts class labels for a factor by using a probability found from the training data. The classifier assumes that all features contribute equally and are independent of each other. This classifier relies on **conditional probability**, or the probability of an event A occurring, given that an event B has occurred:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(B)}{P(A)}$$

In the Naive Bayes setting, the probability of level L for class C (denoted C_L), given feature F, is:

$$P(C_L|F) = \frac{P(F|C_L)P(F)}{P(C_L)}$$

This is generalizable to:

$$P(C_L|F_1, F_2, ..., F_n) = \frac{P(F_1, F_2, ..., F_n|C_L)P(F_1, F_2, ..., F_n)}{P(C_L)} = P(C_L)\prod_{i=1}^{n} P(F_i|C_L)$$

Functions related to running Naive Bayes classification in R are:

Function	Package	Description	Example
`naiveBayes()`	e1071	Train a Naive Bayes classifier.	`naiveBayes(y ~ ., data, laplace = 1)`

Function	Package	Description	Example
`predict()`	stats	Predict values using the trained model.	`m <- naiveBayes(y ~ ., data)` `predict(m, test.data)`
`confusionMatrix()`	caret	Calculate a confusion matrix.	`confusionMatrix(predicted, ground.truth)`