Polynomial Regression

Veerasak Kritsanapraphan

1/17/2020

### Loading Required R packages

library(tidyverse)  
library(modelr)  
library(broom)  
library(dplyr)  
library(ggplot2)

### Preparing the data

We’ll use the Boston data set [in MASS package], for predicting the median house value (mdev), in Boston Suburbs, based on the predictor variable lstat (percentage of lower status of the population).

We’ll randomly split the data into training set (80% for building a predictive model) and test set (20% for evaluating the model).

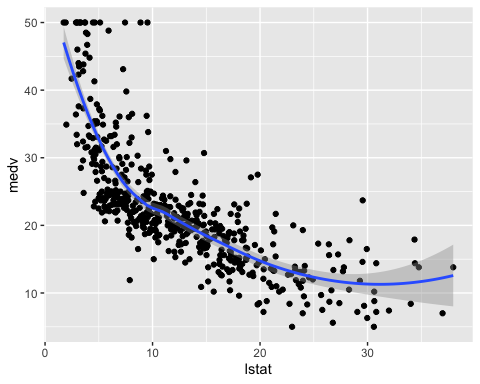
#### Load the data

data("Boston", package = "MASS")

First, visualize the scatter plot of the medv vs lstat variables as follow:

ggplot(Boston, aes(lstat, medv) ) +  
 geom\_point() +  
 stat\_smooth()

## `geom\_smooth()` using method = 'loess' and formula 'y ~ x'



The above scatter plot suggests a non-linear relationship between the two variables

In the following sections, we start by computing linear and non-linear regression models. Next, we’ll compare the different models in order to choose the best one for our data.

## Linear regression

The standard linear regression model equation can be written as medv = b0 + b1\*lstat.

### Compute linear regression model:

### Build the model

model <- lm(medv ~ lstat, data = Boston)

### Model performance

linearmodel <- data.frame(  
 name = 'Linear Regression model',  
 R2 = rsquare(model, data=Boston),  
 RMSE = rmse(model, data=Boston),  
 MAE = mae(model, data=Boston)  
)  
linearmodel

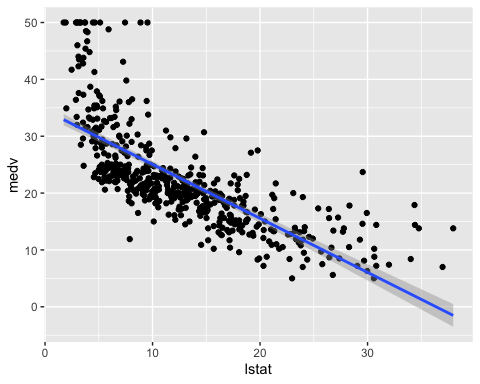
## name R2 RMSE MAE  
## 1 Linear Regression model 0.5441463 6.203464 4.505281

glance(model) %>%  
 dplyr::select(r.squared, adj.r.squared, sigma, AIC, BIC, p.value)

## # A tibble: 1 x 6  
## r.squared adj.r.squared sigma AIC BIC p.value  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.544 0.543 6.22 3289. 3302. 5.08e-88

### Visualize the data:

ggplot(Boston, aes(lstat, medv) ) +  
 geom\_point() +  
 stat\_smooth(method = lm, formula = y ~ x)



## Polynomial regression

The polynomial regression adds polynomial or quadratic terms to the regression equation as follow:

medv=b0+b1∗lstat+b2∗lstat2

In R, to create a predictor x^2 you should use the function I(), as follow: I(x^2). This raise x to the power 2.

The polynomial regression can be computed in R as follow:

lm(medv ~ lstat + I(lstat^2), data = Boston)

##   
## Call:  
## lm(formula = medv ~ lstat + I(lstat^2), data = Boston)  
##   
## Coefficients:  
## (Intercept) lstat I(lstat^2)   
## 42.86201 -2.33282 0.04355

An alternative simple solution is to use this:

lm(medv ~ poly(lstat, 2, raw = TRUE), data = Boston)

##   
## Call:  
## lm(formula = medv ~ poly(lstat, 2, raw = TRUE), data = Boston)  
##   
## Coefficients:  
## (Intercept) poly(lstat, 2, raw = TRUE)1   
## 42.86201 -2.33282   
## poly(lstat, 2, raw = TRUE)2   
## 0.04355

The following example computes a sixfth-order polynomial fit:

lm(medv ~ poly(lstat, 6, raw = TRUE), data = Boston) %>%  
 summary()

##   
## Call:  
## lm(formula = medv ~ poly(lstat, 6, raw = TRUE), data = Boston)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -14.7317 -3.1571 -0.6941 2.0756 26.8994   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 7.304e+01 5.593e+00 13.059 < 2e-16 \*\*\*  
## poly(lstat, 6, raw = TRUE)1 -1.517e+01 2.965e+00 -5.115 4.49e-07 \*\*\*  
## poly(lstat, 6, raw = TRUE)2 1.930e+00 5.713e-01 3.378 0.000788 \*\*\*  
## poly(lstat, 6, raw = TRUE)3 -1.307e-01 5.202e-02 -2.513 0.012295 \*   
## poly(lstat, 6, raw = TRUE)4 4.686e-03 2.407e-03 1.947 0.052066 .   
## poly(lstat, 6, raw = TRUE)5 -8.416e-05 5.450e-05 -1.544 0.123186   
## poly(lstat, 6, raw = TRUE)6 5.974e-07 4.783e-07 1.249 0.212313   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 5.212 on 499 degrees of freedom  
## Multiple R-squared: 0.6827, Adjusted R-squared: 0.6789   
## F-statistic: 178.9 on 6 and 499 DF, p-value: < 2.2e-16

From the output above, it can be seen that polynomial terms beyond the fith order are not significant. So, just create a fifth polynomial regression model as follow:

### Build the fifth polynomial model

model <- lm(medv ~ poly(lstat, 5, raw = TRUE), data = Boston)

### Model performance

polymodel <- data.frame(  
 name = 'Polynomial Regression model',  
 R2 = rsquare(model, data=Boston),  
 RMSE = rmse(model, data=Boston),  
 MAE = mae(model, data=Boston)  
)  
polymodel

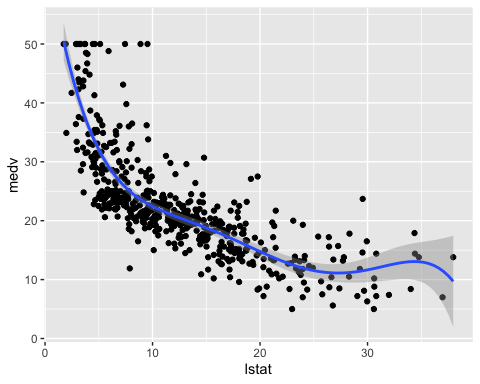
## name R2 RMSE MAE  
## 1 Polynomial Regression model 0.6816897 5.183783 3.732084

glance(model) %>%  
 dplyr::select(r.squared, adj.r.squared, sigma, AIC, BIC, p.value)

## # A tibble: 1 x 6  
## r.squared adj.r.squared sigma AIC BIC p.value  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.682 0.679 5.21 3115. 3145. 8.73e-122

Visualize the fith polynomial regression line as follow:

ggplot(Boston, aes(lstat, medv) ) +  
 geom\_point() +  
 stat\_smooth(method = lm, formula = y ~ poly(x, 5, raw = TRUE))



## Log transformation

When you have a non-linear relationship, you can also try a logarithm transformation of the predictor variables:

### Build the model

model <- lm(medv ~ log(lstat), data = Boston)

### Model performance

logmodel <- data.frame(  
 name = 'Log Transform Regression model',  
 R2 = rsquare(model, data=Boston),  
 RMSE = rmse(model, data=Boston),  
 MAE = mae(model, data=Boston)  
)  
logmodel

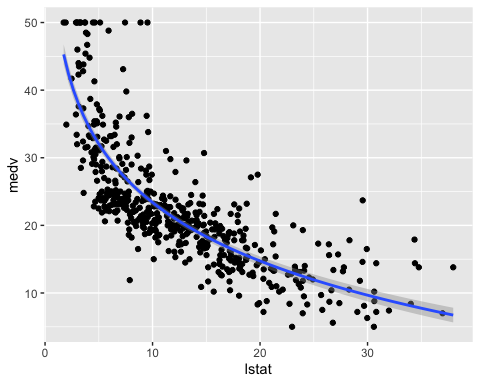
## name R2 RMSE MAE  
## 1 Log Transform Regression model 0.6649462 5.318373 3.88215

glance(model) %>%  
 dplyr::select(r.squared, adj.r.squared, sigma, AIC, BIC, p.value)

## # A tibble: 1 x 6  
## r.squared adj.r.squared sigma AIC BIC p.value  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.665 0.664 5.33 3133. 3146. 9.28e-122

Visualize the data:

ggplot(Boston, aes(lstat, medv) ) +  
 geom\_point() +  
 stat\_smooth(method = lm, formula = y ~ log(x))



## Spline regression

Polynomial regression only captures a certain amount of curvature in a nonlinear relationship. An alternative, and often superior, approach to modeling nonlinear relationships is to use splines.

Splines provide a way to smoothly interpolate between fixed points, called knots. Polynomial regression is computed between knots. In other words, splines are series of polynomial segments strung together, joining at knots.

The R package splines includes the function bs for creating a b-spline term in a regression model.

You need to specify two parameters: the degree of the polynomial and the location of the knots. We’ll place the knots at the lower quartile, the median quartile, and the upper quartile:

knots <- quantile(Boston$lstat, p = c(0.25, 0.5, 0.75))

We’ll create a model using a cubic spline (degree = 3):

library(splines)

#### Build the model

knots <- quantile(Boston$lstat, p = c(0.25, 0.5, 0.75))  
model <- lm (medv ~ bs(lstat, knots = knots), data = Boston)

#### Model performance

splinemodel <- data.frame(  
 name = 'Splines model',  
 R2 = rsquare(model, data=Boston),  
 RMSE = rmse(model, data=Boston),  
 MAE = mae(model, data=Boston)  
)  
splinemodel

## name R2 RMSE MAE  
## 1 Splines model 0.6833446 5.170291 3.704185

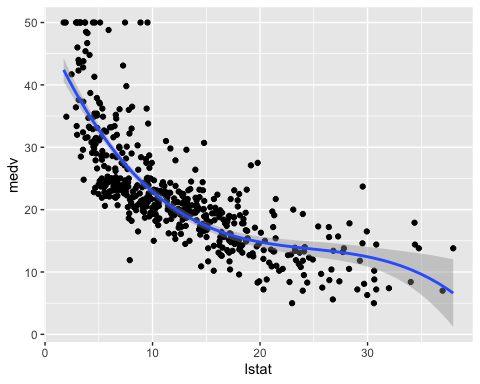
glance(model) %>%  
 dplyr::select(r.squared, adj.r.squared, sigma, AIC, BIC, p.value)

## # A tibble: 1 x 6  
## r.squared adj.r.squared sigma AIC BIC p.value  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 0.683 0.680 5.21 3115. 3148. 3.68e-121

Note that, the coefficients for a spline term are not interpretable.

Visualize the cubic spline as follow:

ggplot(Boston, aes(lstat, medv) ) +  
 geom\_point() +  
 stat\_smooth(method = lm, formula = y ~ splines::bs(x, df = 3))



## Generalized additive models

Once you have detected a non-linear relationship in your data, the polynomial terms may not be flexible enough to capture the relationship, and spline terms require specifying the knots.

Generalized additive models, or GAM, are a technique to automatically fit a spline regression. This can be done using the mgcv R package:

library(mgcv)

#### Build the model

model <- gam(medv ~ s(lstat), data = Boston)

#### Model performance

GAMmodel <- data.frame(  
 name = 'Generalized additive model',  
 R2 = rsquare(model, data=Boston),  
 RMSE = rmse(model, data=Boston),  
 MAE = mae(model, data=Boston)  
)  
GAMmodel

## name R2 RMSE MAE  
## 1 Generalized additive model 0.6849719 5.156988 3.717194

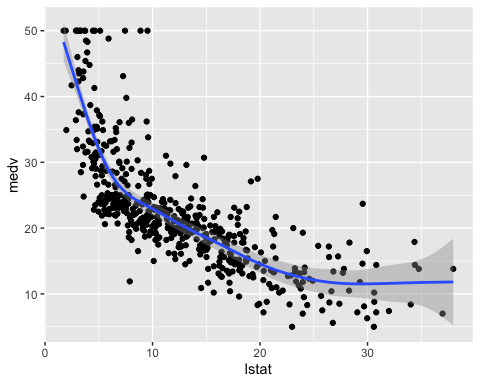
glance(model)

## # A tibble: 1 x 6  
## df logLik AIC BIC deviance df.residual  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 8.40 -1548. 3115. 3155. 13457. 498.

The term s(lstat) tells the gam() function to find the best knots for a spline term.

Visualize the data:

ggplot(Boston, aes(lstat, medv) ) +  
 geom\_point() +  
 stat\_smooth(method = gam, formula = y ~ s(x))



## Comparing the models

From analyzing the RMSE and the R2 metrics of the different models, it can be seen that the polynomial regression, the spline regression and the generalized additive models outperform the linear regression model and the log transformation approaches.

totalmodel <- do.call('rbind', list(linearmodel, polymodel, splinemodel, GAMmodel))  
totalmodel

## name R2 RMSE MAE  
## 1 Linear Regression model 0.5441463 6.203464 4.505281  
## 2 Polynomial Regression model 0.6816897 5.183783 3.732084  
## 3 Splines model 0.6833446 5.170291 3.704185  
## 4 Generalized additive model 0.6849719 5.156988 3.717194