Data621 - Blog5

Amit Kapoor

5/14/2021

Linear Regression

"In statistics, linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables" - Wikipedia

Linear regression does model the relationship between two variables by fitting a linear equation to observed data. The independent variable is considered to be an explanatory variable, and the response variable is considered to be a dependent variable. To start fitting a linear model to observed data, we should first determine if there is a relationship between the variables of interest and they are associated. We could find it through viasualizations like scatter plots between the dependent and predictor variables.

Simple Linear Regression

Lets consider the data collected in pairs (x_1,y_1), (x_2,y_2)...., (x_n,y_n) where X-variable is called explanatory or predictor variable while Y-variable is called response or dependent variable. Simple linear regression is used to model the relationship between two variables yi and xi that can be represented as

$y_i = \beta_0 + \beta_1 \, x_1 + \epsilon_i$

The noise ϵ_i represents that model doesn't fit perfectly with the data, β_0 is intercept and β_1 is slope.

There are four assumptions associated with a linear regression model:

- Linearity: The relationship between X and Y is linear.
- Homoscedasticity: The variance of residual is same for any value of X.
- Independence: Observations are independent of each other.
- Normality: For any fixed value of X, Y is normally distributed

To compute linear regression, `lm` function is used to fit linear models. To look the model, we use `summary` function.

We will use `mtcars` dataset to implement all the models, discussed in this blog. This dataset has below variables.

- mpg Miles/(US) gallon
- cyl Number of cylinders
- disp Displacement (cu.in.)
- hp Gross horsepower
- drat Rear axle ratio
- wt Weight (1000 lbs)
- qsec 1/4 mile time
- vs Engine (0 = V-shaped, 1 = straight)
- am Transmission (0 = automatic, 1 = manual)
- gear Number of forward gears
- carb Number of carburetors

```r
set.seed(317)

# linear model
slr.tune <- lm(mpg ~ wt, data = mtcars)
summary(slr.tune)
```

```
##
## Call:
## lm(formula = mpg ~ wt, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5432 -2.3647 -0.1252  1.4096  6.8727
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
## wt           -5.3445     0.5591  -9.559 1.29e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.046 on 30 degrees of freedom
## Multiple R-squared:  0.7528, Adjusted R-squared:  0.7446
## F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

R^2 of above linear model is 0.75 which means model covers 75% variation of `mpg` is explained by `wt` in this model..

Multiple Linear Regression

In this case, instead of just a single scalar value x, we have now a vector (x_1, x_2 …… , x_p) for every data point i. Here we have n data points, each with p different features. We can represent our input data as X, an x x p matrix where each row corresponds to a data point and each column is a feature. So our linear model can be expressed

$Y = \beta_1 X + \epsilon$

where β is a p element vector of coefficients and ϵ is an n element matrix where each element ϵ_i is normal with mean 0 and variance σ^2.

To implement multiple linear regression, we would use again the same `lm` function with multiple predictors from `mtcars` dataset.

```r
set.seed(317)

# multiple linear regression
mlr.tune <- lm(mpg ~ wt+disp, data = mtcars)
summary(mlr.tune)
```

```
##
## Call:
## lm(formula = mpg ~ wt + disp, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4087 -2.3243 -0.7683  1.7721  6.3484
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.96055    2.16454  16.151 4.91e-16 ***
## wt          -3.35082    1.16413  -2.878  0.00743 **
## disp        -0.01773    0.00919  -1.929  0.06362 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.917 on 29 degrees of freedom
## Multiple R-squared:  0.7809, Adjusted R-squared:  0.7658
## F-statistic: 51.69 on 2 and 29 DF,  p-value: 2.744e-10
```

Partial Least Square

Partial least squares (PLS) is an alternative to ordinary least squares (OLS) regression. It reduces the predictors to a smaller set of uncorrelated components and then performs least squares regression on these components, instead of on the original data. We can use PLS regression when the predictors (independent variables) are highly collinear, or when we have more predictors than observations and ordinary least-squares regression doesn't give the desired results. PLS finds linear combinations of the predictors called components. PLS finds components that attempts to maximally summarize the variation of the predictors while at the same time attempts these components to have maximum correlation with the response.

The `pls` package has functions for PLS model. The `plsr` function implements Partial least squares Regression models.

```
set.seed(317)

# tune pls model
pls.tune <- plsr(mpg ~ disp+hp+drat+wt+qsec, data=mtcars)

summary(pls.tune)
```

```
## Data:    X dimension: 32 5
##  Y dimension: 32 1
## Fit method: kernelpls
## Number of components considered: 5
## TRAINING: % variance explained
##      1 comps  2 comps  3 comps  4 comps  5 comps
## X      92.74    99.99   100.00   100.00   100.00
## mpg    74.54    74.83    77.95    84.88    84.89
```

From the above summary, number of components considered are 5 and it shows the% variation explained.

Penalized Models

The standard linear model (or the ordinary least squares method) performs poorly in a situation, where we have a large multivariate data set containing a number of variables more than the number of observations. A better alternative is the Penalized Regression allowing to create a linear regression model that is penalized for having too many variables in the model and it adds a constraint (coefficient) in the equation. It is also known as regularization methods.

Adding the penalty allows the less contributive variables to have a coefficient close to or equal zero. This shrinkage requires the selection of a tuning parameter (lambda) that determines the amount of shrinkage.

Ridge

Ridge regression shrinks the regression coefficients so predictor variables that contributes less to the outcome, have their coefficients close to zero. The shrinkage of the coefficients is achieved by penalizing the regression model with a penalty term called L2(second order penalty), which is the sum of the squared coefficients.

When the model overfits the data or in case collinearity, we may need to control the magnitude of linear regression parameter estimates to reduce the SSE. Controlling (or regularizing) the parameter estimates can be achieved by adding a penalty to the SSE if the estimates become large. Ridge regression adds a penalty on the sum of the squared regression parameters:

$$\text{SSE}_{L_2} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{P} \beta_j^2$$

To tune over the penalty, `train` can be used with a `ridge` method.

```
set.seed(317)

y <- mtcars %>% select(mpg) %>% as.matrix()
X <- mtcars %>% select(disp,hp,drat,wt,qsec) %>% as.matrix()

# tune ridge model
ridge.fit <- train(mpg ~ .,
                   data = cbind(y, X),
                   method="ridge",
                   metric="Rsquared",
                   tuneGrid = data.frame(lambda=seq(0,1,by=0.1)),
                   trControl=trainControl(method = "cv",number=5),
                   preProcess=c("center", "scale")
                  )

ridge.fit
```

```
## Ridge Regression
##
## 32 samples
##  5 predictor
##
## Pre-processing: centered (5), scaled (5)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 27, 25, 24, 26, 26
## Resampling results across tuning parameters:
##
##   lambda  RMSE      Rsquared   MAE
##   0.0     2.960116  0.8909997  2.463497
##   0.1     2.959709  0.8786805  2.567977
##   0.2     3.043585  0.8734554  2.696009
##   0.3     3.155250  0.8710062  2.817039
##   0.4     3.285646  0.8695794  2.932687
##   0.5     3.429025  0.8686302  3.043156
```

```
##   0.6     3.581158  0.8679417  3.148605
##   0.7     3.738887  0.8674115  3.270851
##   0.8     3.899840  0.8669854  3.402188
##   0.9     4.062221  0.8666319  3.545806
##   1.0     4.224653  0.8663315  3.690889
##
## Rsquared was used to select the optimal model using the largest value.
## The final value used for the model was lambda = 0.
```

Lasso

Lasso stands for "Least Absolute Shrinkage and Selection Operator". It shrinks the regression coefficients toward zero by penalizing the regression model with a penalty term called L1, which is the sum of the absolute coefficients. In the case of lasso regression, the penalty has the effect of forcing some of the coefficient estimates having minor contribution to the model, to be exactly equal to zero. Lasso regression adds a penalty on the sum of absolute regression parameters.

$$\text{SSE}_{L_1} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{P} |\beta_j|$$

To tune over the penalty, `train` can be used with a `lasso` method.

```
set.seed(317)
# tune lasso model
lasso.fit <- train(mpg ~ .,
                   data = cbind(y, X),
                   method="lasso",
                   metric="Rsquared",
                   tuneGrid = data.frame(fraction=seq(0,1,by=0.1)),
                   trControl=trainControl(method = "cv",number=5),
                   preProcess=c("center", "scale")
                  )

lasso.fit
```

```
## The lasso
##
## 32 samples
##  5 predictor
##
## Pre-processing: centered (5), scaled (5)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 27, 25, 24, 26, 26
## Resampling results across tuning parameters:
##
##   fraction  RMSE      Rsquared   MAE
##   0.0       6.092495        NaN  4.790702
##   0.1       5.258520  0.7926746  4.085123
##   0.2       4.506226  0.8148864  3.472276
##   0.3       3.831941  0.8329375  2.935417
```

```
##   0.4       3.286463  0.8428401  2.564175
##   0.5       2.963974  0.8608627  2.308459
##   0.6       2.896498  0.8746895  2.275238
##   0.7       2.982438  0.8798416  2.416057
##   0.8       2.954881  0.8866463  2.397585
##   0.9       2.947916  0.8898526  2.430541
##   1.0       2.960116  0.8909997  2.463497
##
## Rsquared was used to select the optimal model using the largest value.
## The final value used for the model was fraction = 1.
```

Elastic Net

Elastic Net produces a regression model that is penalized with both the L1-norm and L2-norm. The consequence of this is to effectively shrink coefficients (like in ridge regression) and to set some coefficients to zero (as in LASSO). The main advantage of elastic net model is that it enables effective regularization by having the ridge-type penalty along with the feature selection quality of the lasso penalty.

$$\text{SSE}_{\text{Enet}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{P} \beta_j^2 + \lambda_2 \sum_{j=1}^{P} |\beta_j|$$

The `elastic net` penalty is controlled by α and bridges the gap between `lasso` regression (α=1), the default) and `ridge` regression (α=0)

```
set.seed(317)
# tune enet model
enet.fit <- train(mpg ~ .,
                  data = cbind(y, X),
                  method="enet",
                  metric="Rsquared",
                  tuneGrid = expand.grid(fraction=seq(0,1,by=0.5), lambda=seq(0,1,by=0.1)),
                  trControl=trainControl(method = "cv",number=5),
                  preProcess=c("center", "scale")
                 )

enet.fit
```

```
## Elasticnet
##
## 32 samples
##  5 predictor
##
## Pre-processing: centered (5), scaled (5)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 27, 25, 24, 26, 26
## Resampling results across tuning parameters:
##
##   lambda  fraction  RMSE      Rsquared   MAE
##   0.0     0.0       6.092495        NaN  4.790702
##   0.0     0.5       2.963974  0.8608627  2.308459
```

```
##   0.0     1.0       2.960116  0.8909997  2.463497
##   0.1     0.0       6.092495        NaN  4.790702
##   0.1     0.5       3.410338  0.8434706  2.738572
##   0.1     1.0       2.959709  0.8786805  2.567977
##   0.2     0.0       6.092495        NaN  4.790702
##   0.2     0.5       3.304626  0.8442705  2.673975
##   0.2     1.0       3.043585  0.8734554  2.696009
##   0.3     0.0       6.092495        NaN  4.790702
##   0.3     0.5       3.212594  0.8467468  2.610914
##   0.3     1.0       3.155250  0.8710062  2.817039
##   0.4     0.0       6.092495        NaN  4.790702
##   0.4     0.5       3.134282  0.8503048  2.556383
##   0.4     1.0       3.285646  0.8695794  2.932687
##   0.5     0.0       6.092495        NaN  4.790702
##   0.5     0.5       3.064486  0.8532132  2.503132
##   0.5     1.0       3.429025  0.8686302  3.043156
##   0.6     0.0       6.092495        NaN  4.790702
##   0.6     0.5       3.004086  0.8553740  2.458839
##   0.6     1.0       3.581158  0.8679417  3.148605
##   0.7     0.0       6.092495        NaN  4.790702
##   0.7     0.5       2.952792  0.8570378  2.416551
##   0.7     1.0       3.738887  0.8674115  3.270851
##   0.8     0.0       6.092495        NaN  4.790702
##   0.8     0.5       2.910074  0.8583557  2.377308
##   0.8     1.0       3.899840  0.8669854  3.402188
##   0.9     0.0       6.092495        NaN  4.790702
##   0.9     0.5       2.875276  0.8594240  2.344480
##   0.9     1.0       4.062221  0.8666319  3.545806
##   1.0     0.0       6.092495        NaN  4.790702
##   1.0     0.5       2.847685  0.8603063  2.320124
##   1.0     1.0       4.224653  0.8663315  3.690889
##
## Rsquared was used to select the optimal model using the largest value.
## The final values used for the model were fraction = 1 and lambda = 0.
```

References

- Applied Predictive Modeling by Max Kuhn and Kjell Johnson
- https://glmnet.stanford.edu/articles/glmnet.html