ML_1 Model Summary

Ian Lawson & Carrington Metts

11/11/2020

Part 1: Prediction

Simple Linear Regression

General Description

Simple linear regression is used to predict linear data. The data need to follow a linear pattern in order to be useful. The model give the equation for a line ie y = m*x + b where m is the slope and b is the y intercept. This is the most basic form of regression.

When to Use It

Use simple linear regression when you data appears to be linear (looks like a line) and you have 1 independent variable and 1 dependent variable.

Modeling Functions

linear model function: model<-lm(dependent_var~independent_var, data=your_data)

Modeling Function Example

```r
#load data
data('longley')
#create model
lmEcon<-lm(Employed~Population,longley)
```

Assumptions and Tests

In summary, the assumptions for linear regression are: 1. Linearity of the data 2. Significance of the model (p values and R^2) 3. Lack of highly influential outliers 4. Normality of residuals 5. Homoscedasticity

Linearity of Data First it's a good idea just to plot the data to see if it is linear. Use: plot(dependent_var~independent_var, your_data)

Significance of Model After making your model, look at the summary of the model. First, the p-values needs to be below 0.05 for the model to pass. If this is not met, the model need to be adjusted (remove outliers or don't use this type of model). Next, look at the R^2 values. They show the percentage of the variability of the data that is explained by the relationship of the dependent and independent variables. Generally, the higher the R^2 value, the better. Adjusted R-squared is used when the data is from a small sample size. With large datasets, the R-squared and adjusted R-squared should be nearly equal.

Influential Outliers Next, plot the linear model using plot(model). The first and third plots (Residuals vs Fitted and Scale-Location) ideally will not have any type of pattern. They will just look like a random set of points. Particularly, funnel shapes are a bad indicator. (In particular, a funnel shape may indicate heteroscedasticity.) The second plot, Normal Q-Q should have points that follow the line on the plot. Points that deviate can be considered outliers and if there is to much deviation, linear regression may not be the optimal model. The last plot shows any Cook's Distance issues. Points that fall outside of the limits shown in the graph are highly influential and can be considered outliers. These points should be investigated.

Cook's distance should be tested for next. This is done by using: plot(model, which = c(4)) Cook's distance is violated when values are greater than 4/N where N is the number of data points. This is not a hard rule, and plot four above can also be used. Any violating data points should be investigated.

Normality of Residuals A histogram of the residuals should be plotted to check for normality. If the histogram appears to be normal, then it passes. If not, outliers should be investigated and/or a new model should be considered. This histogram can be made using: hist(model$residuals)

Homoscedasticity Last, homoscedasticity should be checked using BP test. If the model passes this test (p-value is less than 0.05), then it is heteroscedastic and the model should be reconsidered. To run this test use: lmtest::bptest(model)

In many cases, heteroscedasticity can be fixed by taking the log of the dependent variable. Use: model <- lm(log(dependent_var)~independent_var, data=your_data) If this transformation is done, all other assumptions must be rechecked with the new model.

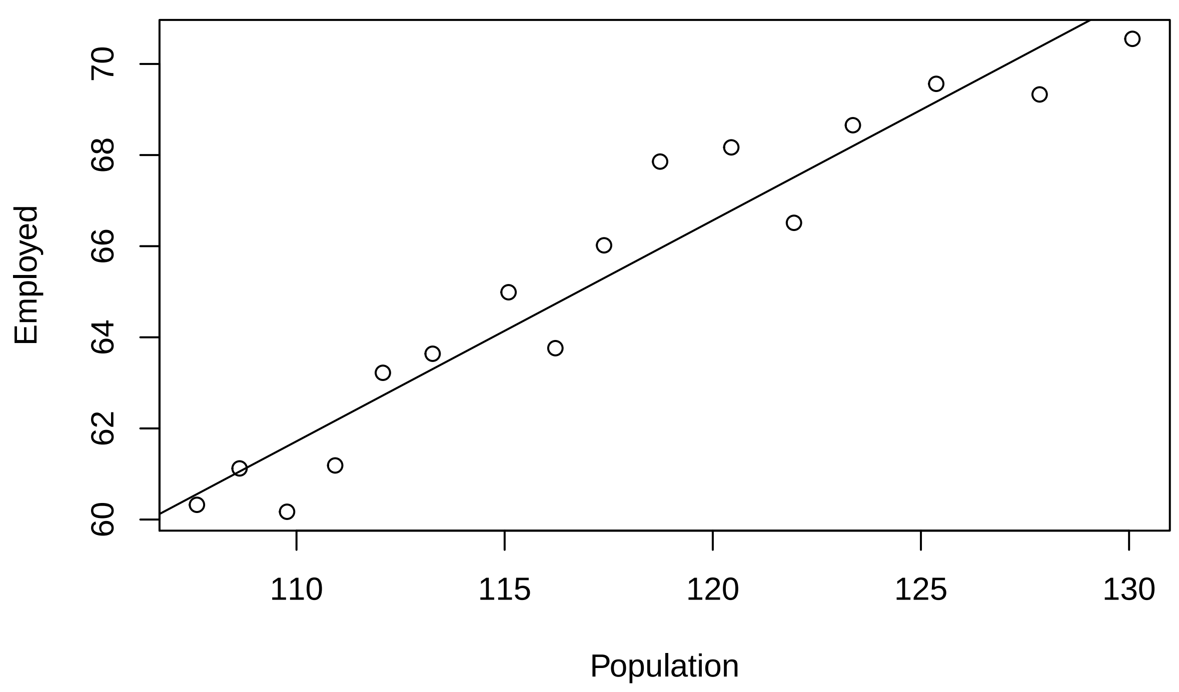
If these test and assumptions are not met, investigate outliers. If outliers cannot be removed, seek another type of model for the data.

Assumptions and Tests Example

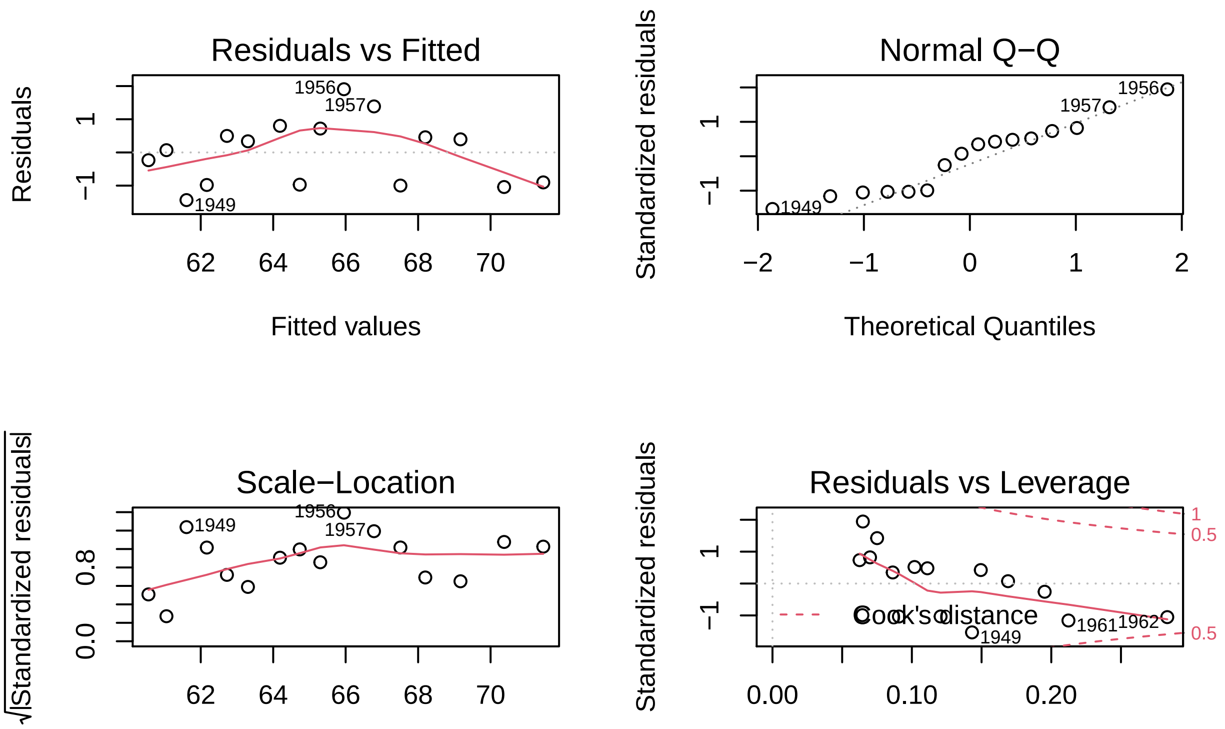
```r
#get the summary
summary(lmEcon)
```

```
##
## Call:
## lm(formula = Employed ~ Population, data = longley)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.4362 -0.9740  0.2021  0.5531  1.9048
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   8.3807     4.4224   1.895   0.0789 .
## Population    0.4849     0.0376  12.896 3.69e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.013 on 14 degrees of freedom
## Multiple R-squared:  0.9224, Adjusted R-squared:  0.9168
## F-statistic: 166.3 on 1 and 14 DF,  p-value: 3.693e-09
```

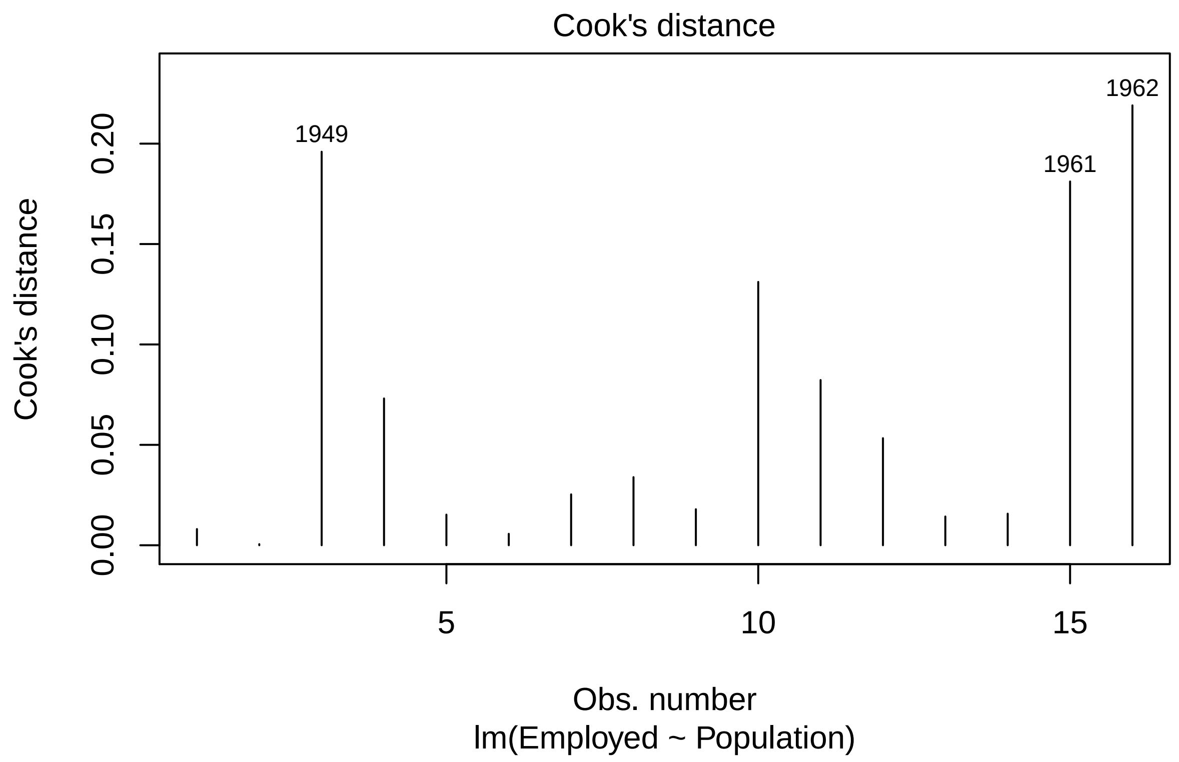
```r
#plot the data
plot(Employed~Population,longley)
#add the regression line
abline(lmEcon)
```



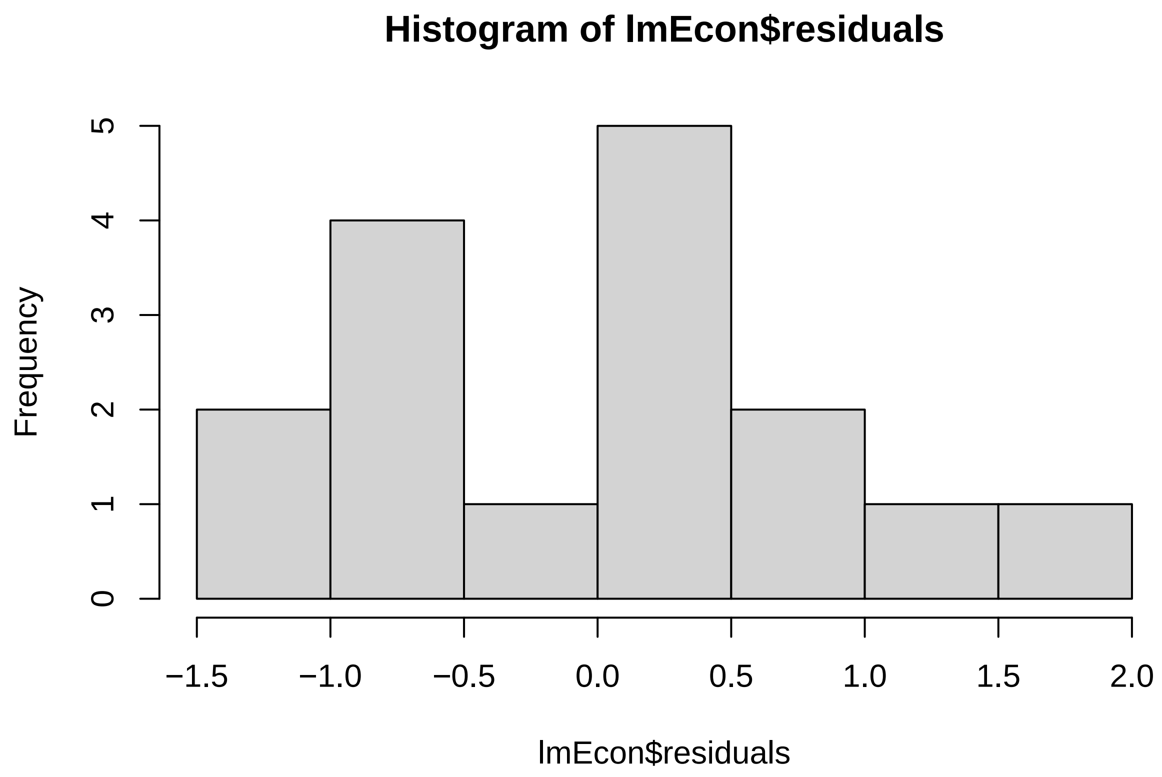
```
#look at linear model plots
par(mfrow=c(2,2)) ## optional way to view charts
plot(lmEcon)
```



```r
par(mfrow=c(1,1))
#plot cooks distance in better method
plot(lmEcon,which = c(4))
```



```r
#test normality of errors
hist(lmEcon$residuals)

#run test for homoscedasticity
lmtest::bptest(lmEcon)
```



```
##
##  studentized Breusch-Pagan test
##
## data:  lmEcon
## BP = 0.23763, df = 1, p-value = 0.6259
```

Outlier Investigation

If any outliers exist, they should be investigated. If it appears that the data point is an abnormality and would not help in predicting future data, the point may be removed. Outliers should be removed one at a time. Each time one is removed, the entire model should be rerun and all the assumptions and tests should be redone. If all the tests and assumptions pass, verify that the model itself improves. The model's overall p-value should decrease and/or r-squared should increase. If this does not happen, if the value changes are very small, or if the model gets worse, it is best to leave them in the model.

Multiple Linear Regression

General Description

Multiple linear regression is used to predict linear data. The data need to follow a linear pattern in order to be useful. The model give the equation for a line ie y = m1*x1 + m2*x2 ... mn*xn + b where n is the number of independent variables, m is the slope, and b is the y intercept.

When to Use It

Use multiple linear regression when you data appears to be linear and there are multiple independent variables.

Modeling Functions

linear model function: model<-lm(dependent_var~independent_var1+independent_var2+…+independent_varn, data=your_data)

Modeling Function Example

```r
#load data
adv<-read.csv('Advertising.csv')

#build regression object
lmAdv<-lm(sales~TV + radio,data = adv)
#note: ~. can be used to represent all other variables instead of typing them out
```

Assumptions and Tests

In summary, the assumptions for multiple linear regression are: 1. Linearity of data 2. Significance of the model 3. Lack of highly influential outliers 4. Normality of residuals 5. Homoscedasticity 6. Multicollinearity

Linearity of Data First it's a good idea just to plot the data to see if it is linear. Use: library(scatterplot3d) scatter<-scatterplot3d(your data[,c(columns_you_want)],color = 'blue',angle = 50,type = 'h') scatter$plane3d(model) This technique only really works with 2 independent variables, since it's notoriously difficult to perceive more than 3 spatial dimensions.

Significance of Model After making your model, look at the summary of the model. First, the p-values needs to be below 0.05 for the model to pass. If this is not met, the model need to be adjusted. To adjust, try removing the independent variable with the highest p-value. Rerun the model and try again until all p-values are acceptable. Next, look at the R^2 values. They show the percentage of the variability of the data that is explained by the relationship of the dependent and independent variables. Generally, the higher the R^2 value, the better. Adjusted R-squared is used when the data is from a small sample size and the the other is used with lots of data.

Highly Influential Outliers Next, plot the linear model using plot(model). The first and third plots (Residuals vs Fitted and Scale-Location) ideally will not have any type of pattern. They will just look like a random set of points. Particularly, funnel shapes are a bad indicator. The second plot, Normal Q-Q should have points that follow the line on the plot. Points that deviate can be considered outliers and if there is to much deviation, linear regression may not be the optimal model. The last plot shows any Cook's Distance issues. Points that fall outside of the limits shown in the graph are highly influential and can be considered outliers. These points should be investigated.

Cook's distance should be tested for next. This is done by using: plot(model, which = c(4)) Cook's distance is violated when values are greater than 4/N where N is the number of data points. This is not a hard rule, and plot four above can also be used. Any violating data points should be investigated.

Normality of Residuals A histogram of the residuals should be plotted to check for normality. If the histogram appears to be normal and centered at 0, then it passes. If not, outliers should be investigated and/or a new model should be considered. This histogram can be made using: hist(model$residuals)Themeanoftheresidualscanbecheckedusing : mean(model$residuals)

Homoscedasticity Homoscedasticity should be checked using BP test. If the model passes this test (p-value is less than 0.05), then it is heteroscedastic and the model should be reconsidered. To run this test use: lmtest::bptest(model)

Multicollinearity Last, VIF should be tested. Any scores that the test returns that are higher than 10 fail the model. This can be tested for using: car::vif(model)

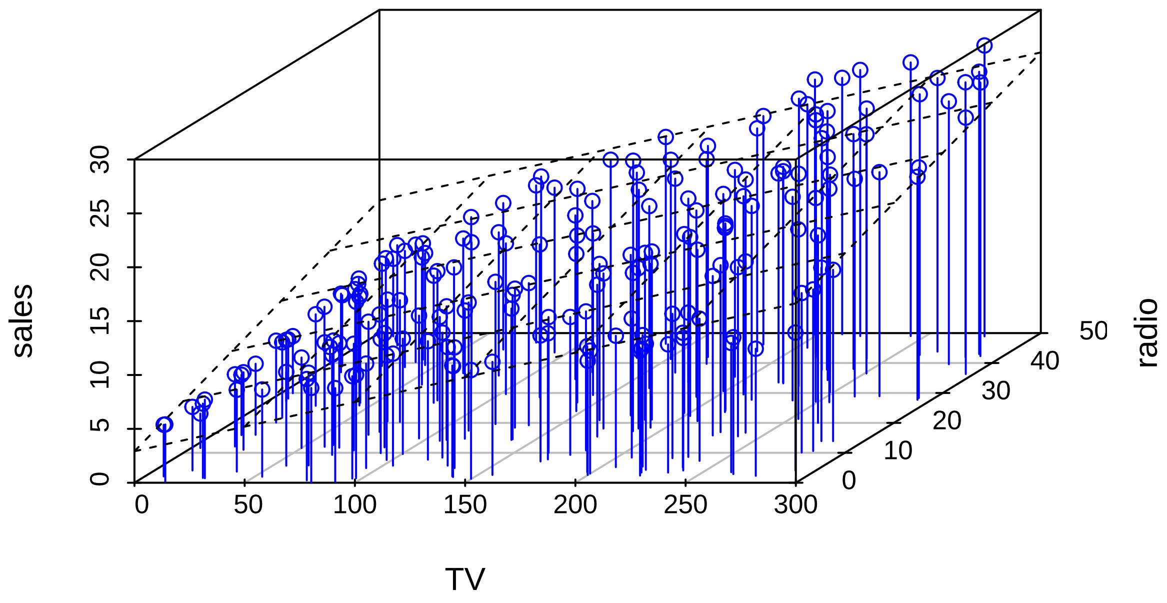
If these test and assumptions are not met, investigate independent variables and outliers. Remove the independent variables one by one until the model passes. If that does not work, try removing outliers one by one. If outliers cannot be removed, seek another type of model for the data.

Assumptions and Tests Example

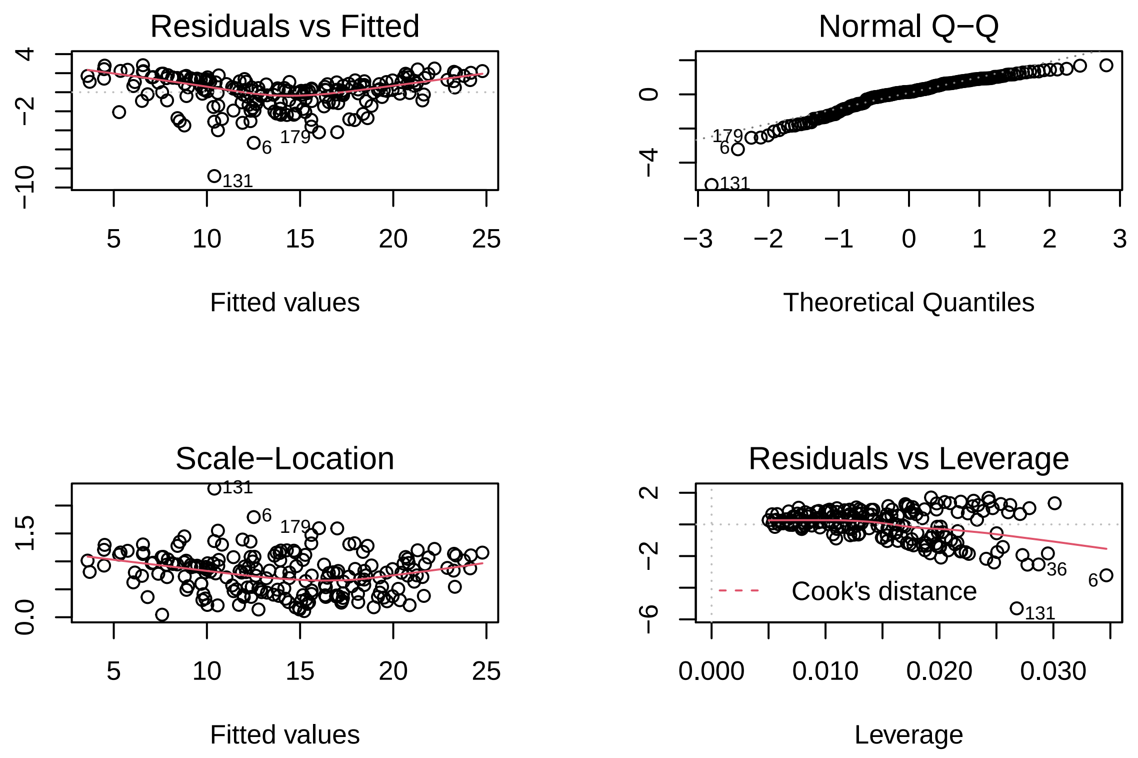
```
#get the summary
summary(lmAdv)
```

```
##
## Call:
## lm(formula = sales ~ TV + radio, data = adv)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.7977 -0.8752  0.2422  1.1708  2.8328
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.92110    0.29449   9.919   <2e-16 ***
## TV           0.04575    0.00139  32.909   <2e-16 ***
## radio        0.18799    0.00804  23.382   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.681 on 197 degrees of freedom
## Multiple R-squared:  0.8972, Adjusted R-squared:  0.8962
## F-statistic: 859.6 on 2 and 197 DF,  p-value: < 2.2e-16
```

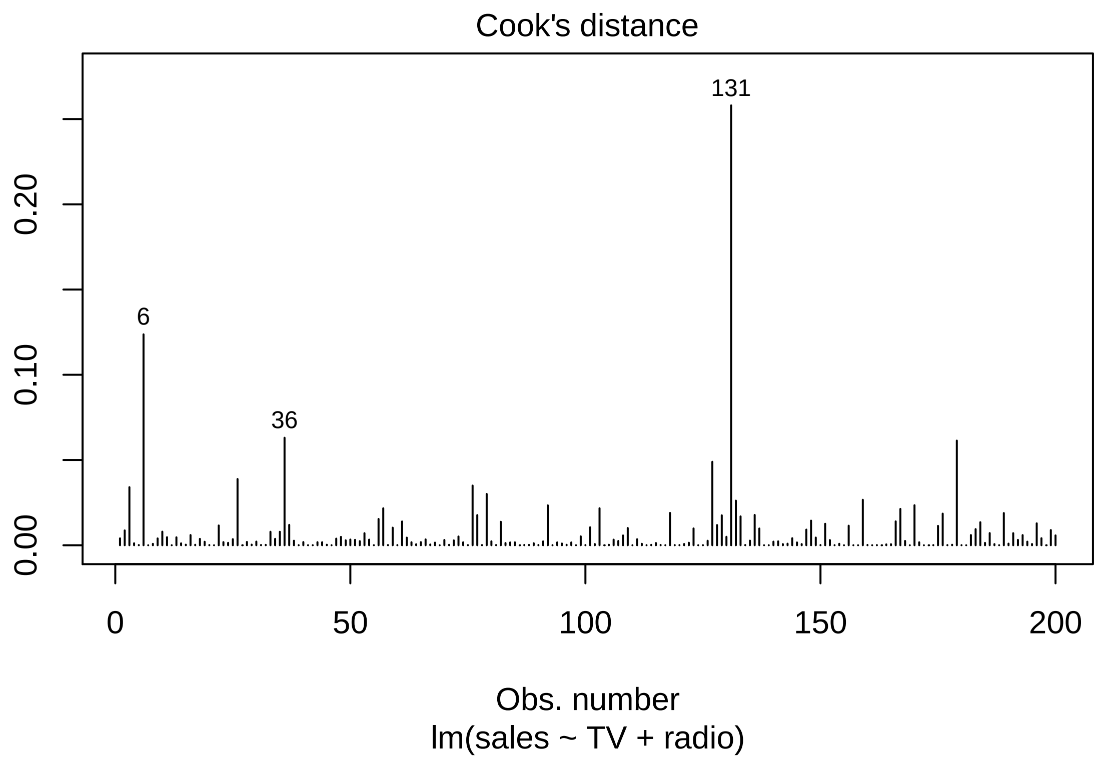
```
#plot the data
library(scatterplot3d)
scatter<-scatterplot3d(adv[,c(2,3,5)],color = 'blue',angle = 50,type = 'h')
#add the regression plane
scatter$plane3d(lmAdv)
```



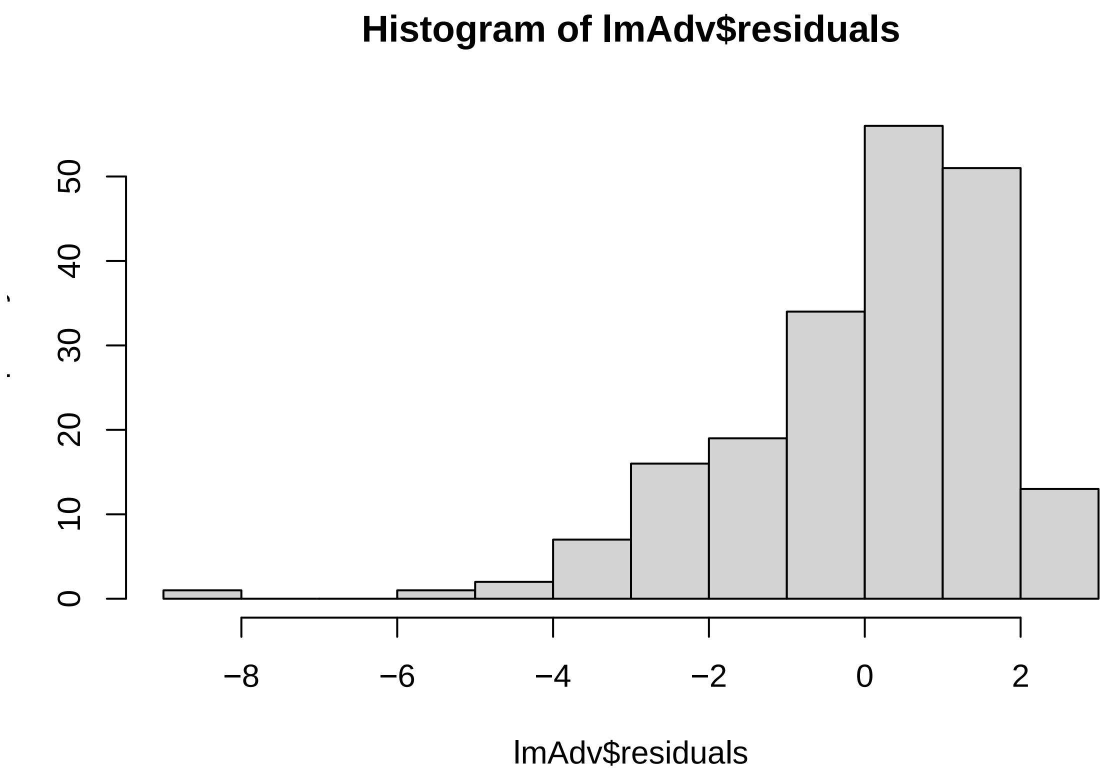
```r
#look at linear model plots
par(mfrow=c(2,2)) ## optional way to view charts
plot(lmAdv)
```



```r
par(mfrow=c(1,1))
#plot cooks distance in better method
plot(lmAdv,which = c(4))
```



```
#test normality of errors
hist(lmAdv$residuals)
```



```r
#run test for homoscedasticity
lmtest::bptest(lmEcon)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  lmEcon
## BP = 0.23763, df = 1, p-value = 0.6259
```

```r
#test VIF
car::vif(lmAdv)
```

```
##       TV    radio
## 1.003013 1.003013
```

Outlier Investigation

If any outliers exist, they should be investigated. If it appears that the data point is an abnormality and would not help in predicting future data, the point may be removed. Outliers should be removed one at a time. Each time one is removed, the entire model should be rerun and all the assumptions and tests should be redone. If all the tests and assumptions pass, verify that the model itself improves. The model's overall p-value should decrease and/or r-squared should increase. If this does not happen, if the value changes are very small, or if the model gets worse, it is best to leave them in the model.

Linear Regression with Interaction

General Description

Linear regression with interaction is used to predict linear data. The data need to follow a linear pattern in order to be useful. The model give the equation for a line ie y = m1*x1*x2 + m2*x1 + m3*x2 . . . + b. This differs from the cases above because the interacting terms are multiplied by each other and also have their own slope associated with them.

When to Use It

Use linear regression with interaction when you data appear to be linear, you have multiple independent variables, and it looks like variables may interact.

Modeling Functions

linear model function: model<-lm(dependent_var~interact_var1*interact_var2…, data=your_data)

Modeling Function Example

```r
#load data
can<-read.csv('cancer_reg.csv')
#build the model
lmcancer3<-lm(target_deathrate~incidencerate * pctmarriedhouseholds, data = can)
summary(lmcancer3)
```

```
##
## Call:
## lm(formula = target_deathrate ~ incidencerate * pctmarriedhouseholds,
##     data = can)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -130.513  -15.788   -1.751   15.397  111.087
##
## Coefficients:
##                                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)                        190.958463  24.745660   7.717 1.61e-14 ***
## incidencerate                        0.084623   0.054110   1.564   0.1179
## pctmarriedhouseholds                -2.096859   0.481679  -4.353 1.39e-05 ***
## incidencerate:pctmarriedhouseholds   0.002497   0.001059   2.358   0.0185 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.96 on 3043 degrees of freedom
## Multiple R-squared:  0.2551, Adjusted R-squared:  0.2544
## F-statistic: 347.5 on 3 and 3043 DF,  p-value: < 2.2e-16
```

Assumptions and Tests

The assumptions and tests for linear regression with interaction are exactly the same as the assumptions and test for multiple linear regression. See the above sections for this information.

Nonlinear Transformation Regression

General Description

Nonlinear transformation regression is used to transform nonlinear variables into linear variables so that linear regression can be preformed. The equation for this follows what every polynomial equation best fits the data. The transformation can take place on both independent and dependent variables. It usually transforms polynomials or exponential data.

When to Use It

Use this type of regression when the data appears to be nonlinear. This can be seen when graphing the data if it is within visual dimensions.

Modeling Functions

linear model function: model<-lm(dependent_var~poly(interact_var,polynomial_degree), data=your_data) or model<-lm(log(dependent_var)~log(interact_var), data=your_data) or model<-lm(dependent_var~log(interact_var), data=your_data)

Modeling Function Example

```
#load data
lifeexp<-read.csv('lifeexp2015.csv')
lifeexp <- na.omit(lifeexp)

#build the model
#model quad
lmlifeQuad<-lm(Life.expectancy~poly(BMI,2),data=lifeexp)
summary(lmlifeQuad)
```

```
##
## Call:
## lm(formula = Life.expectancy ~ poly(BMI, 2), data = lifeexp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -21.1645  -3.8000   0.6754   4.0858  16.2437
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    70.7415     0.5312 133.161  < 2e-16 ***
## poly(BMI, 2)1  49.5027     6.0572   8.173 2.63e-13 ***
## poly(BMI, 2)2  33.7718     6.0572   5.575 1.42e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.057 on 127 degrees of freedom
## Multiple R-squared:  0.4352, Adjusted R-squared:  0.4264
## F-statistic: 48.94 on 2 and 127 DF,  p-value: < 2.2e-16
```

```
#cubic model
lmlifeCube<-lm(Life.expectancy~poly(BMI,3),data=lifeexp)
summary(lmlifeQuad)
```

```
##
```

```
## Call:
## lm(formula = Life.expectancy ~ poly(BMI, 2), data = lifeexp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -21.1645  -3.8000   0.6754   4.0858  16.2437
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    70.7415     0.5312 133.161  < 2e-16 ***
## poly(BMI, 2)1  49.5027     6.0572   8.173 2.63e-13 ***
## poly(BMI, 2)2  33.7718     6.0572   5.575 1.42e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.057 on 127 degrees of freedom
## Multiple R-squared:  0.4352, Adjusted R-squared:  0.4264
## F-statistic: 48.94 on 2 and 127 DF,  p-value: < 2.2e-16
```

```
#quartic model
lmlifeQuart<-lm(Life.expectancy~poly(BMI,4),data=lifeexp)
summary(lmlifeQuad)
```

```
##
## Call:
## lm(formula = Life.expectancy ~ poly(BMI, 2), data = lifeexp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -21.1645  -3.8000   0.6754   4.0858  16.2437
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    70.7415     0.5312 133.161  < 2e-16 ***
## poly(BMI, 2)1  49.5027     6.0572   8.173 2.63e-13 ***
## poly(BMI, 2)2  33.7718     6.0572   5.575 1.42e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.057 on 127 degrees of freedom
## Multiple R-squared:  0.4352, Adjusted R-squared:  0.4264
## F-statistic: 48.94 on 2 and 127 DF,  p-value: < 2.2e-16
```

```
#log independent model
lmlifelog_i<-lm(Life.expectancy~log(BMI),data=lifeexp)
summary(lmlifelog_i)
```

```
##
## Call:
## lm(formula = Life.expectancy ~ log(BMI), data = lifeexp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.7259  -5.2462   0.9436   4.8373  20.0964
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  58.3484     3.0030   19.43  < 2e-16 ***
## log(BMI)      3.5615     0.8419    4.23 4.41e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.52 on 128 degrees of freedom
## Multiple R-squared:  0.1227, Adjusted R-squared:  0.1158
## F-statistic: 17.89 on 1 and 128 DF,  p-value: 4.413e-05
```

```
#log both model
lmlifelog_b<-lm(log(Life.expectancy)~log(BMI),data=lifeexp)
summary(lmlifelog_b)
```

```
##
## Call:
## lm(formula = log(Life.expectancy) ~ log(BMI), data = lifeexp)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.30604 -0.07006  0.01861  0.07166  0.27370
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.07527    0.04424  92.110  < 2e-16 ***
## log(BMI)     0.05090    0.01240   4.103 7.21e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1108 on 128 degrees of freedom
## Multiple R-squared:  0.1162, Adjusted R-squared:  0.1093
## F-statistic: 16.84 on 1 and 128 DF,  p-value: 7.207e-05
```

Assumptions and Tests

The assumptions and tests for nonlinear regression are exactly the same as the assumptions and test for multiple linear regression. See the above sections for this information.

Part 2: Classification

The second part of the course focuses on classification problems. That is, these models allow you to predict the category of a given observation based on past observations. For classification problems, the dependent variables (and many of the independent variables) are categorical instead of continuous. Classification techniques can also be used to predict the probability that a given observation belongs in each of the available categories.

Training and Testing Sets

Classification models are generally created using a fraction of the dataset (the training set) and the accuracy is assessed using the remaining data (the test set). The createDataPartition function in the caret library is used to split the data.

Testing and Training Example

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.3

## Loading required package: lattice

## Loading required package: ggplot2
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 4.0.3

## Loading required package: tibble

## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```r
data(wine)
divideData <- wine$Type %>% createDataPartition(p=.8,list = FALSE)
train <- wine[divideData, ]
test <- wine[-divideData, ]
```

The model can then be created using data=train. The accuracy can be assessed with the test set.

Logistic Regression

General Description

Logistic regression is the simplest form of classification, in which one or more observations are used to predict a binary categorical outcome. For example, banks may use logistic regression to predict whether a customer will default on a loan. The equation describing probability is: b0 + b1*x = log((p(X)/(1-p(X)))), where p(X) = Pr(Y=1 | X)

When to use it

Logistic regression should be used when the dependent variable is binary categorical (that is, yes or no). The independent variable(s) may be categorical or continuous.

Modeling Functions

To create a logistic regression model: model <- glm(dependent~independents, family=binomial, data=train)

To get the coefficients: exp(coef(model))

To find probabilities of falling into a category: probs <- predict(model, test, type='response') preds <- ifelse(probs>0.5, 'Yes', 'No')

Modeling Function Example

```
library(mlbench)
```

```
## Warning: package 'mlbench' was built under R version 4.0.3
```

```
data('PimaIndiansDiabetes2')
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
divideData <- createDataPartition(PimaIndiansDiabetes2$diabetes, p=.8, list=FALSE)
train <- PimaIndiansDiabetes2[divideData, ]
test <- PimaIndiansDiabetes2[-divideData, ]

model <- glm(diabetes~., data=train, family=binomial)

probabilities <- predict(model, test, type='response')
pred <- ifelse(probabilities>0.5, 'pos', 'neg')
mean(pred==test$diabetes) #testing accuracy rate
```

```
## [1] 0.7692308
```

```
mean(pred!=test$diabetes) #testing error rate
```

```
## [1] 0.2307692
```

```
table(pred, test$diabetes)
```

```
##
## pred  neg pos
##   neg  46  12
##   pos   6  14
```

Assumptions and Tests

In summary, the assumptions for logistic regression are: 1. Linearity of the logit 2. Absence of multicollinearity (with multiple independent variables) 3. Lack of strongly influential outliers 4. Independence of errors 5. Significant Observations When making models, it is best to create the model with the entire dataset, test for assumptions, then split the dataset into testing and training groups and determine the accuracy.

Linearity of the Logit The plot of the dependent variable versus the logit should be linear. In other words, the interaction between each independent numeric variable and its log should be insignificant.

Absence of multicollinearity Just like in multiple linear regression, use: car::vif(model) If any scores are greater than 10, there are some collinear variables and the assumption is not passed. Try removing some of the offending variables and rerunning the model.

Lack of Strongly Influential Outliers As in linear regression, points with a very high Cook's distance should be removed.

Independence of Errors This assumption is violated if there are repeated observations in the dataset.

Significant Observations

When looking at a summary of the model, all independent variables should have p values less than 0.05. If not, try removing that variable and recalculating the error rate. NOTE: insignificant observations may be left in the final model if that yields a better error rate.

Assumptions and Tests Example

```r
library(tidyverse)
```

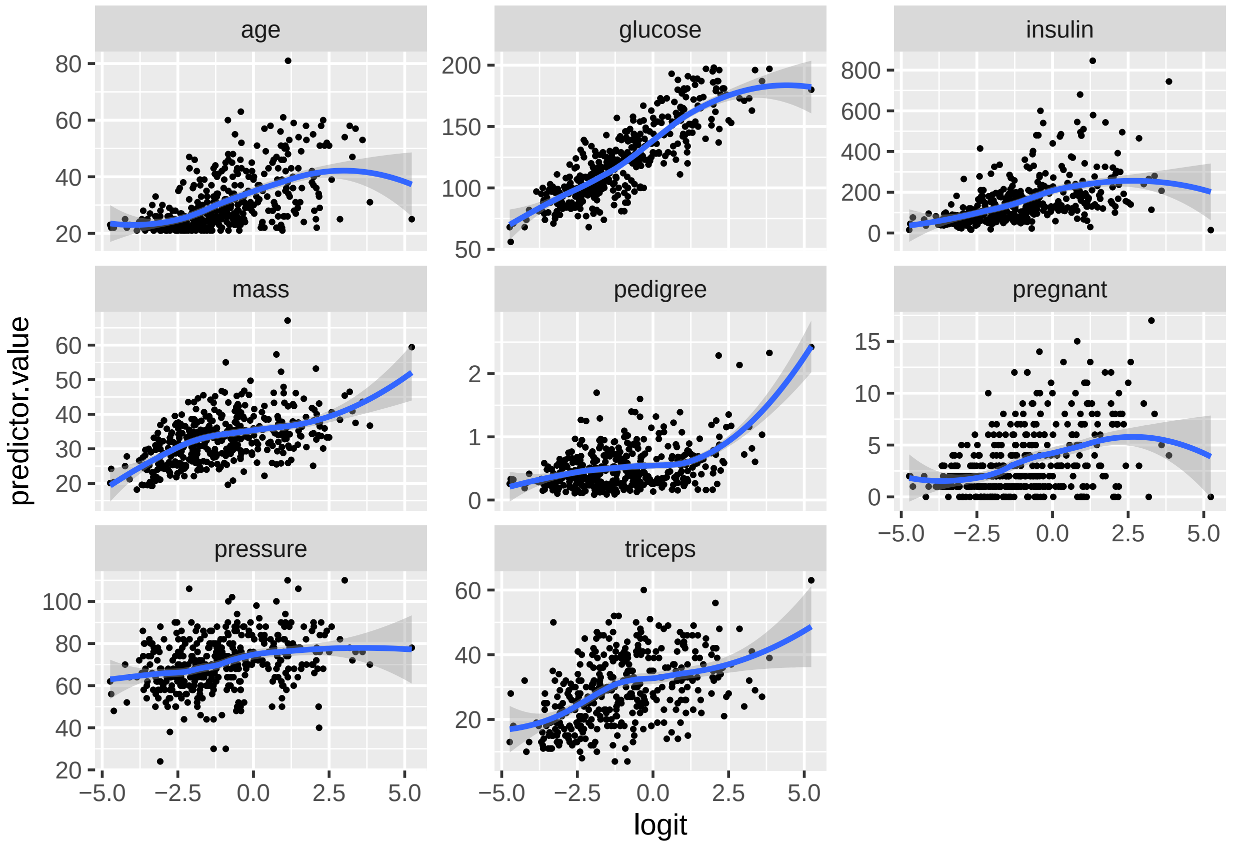
```
## -- Attaching packages ------------------------------------------------------------ tidyverse 1.3.0 --

## v tidyr   1.1.1     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## v purrr   0.3.4

## -- Conflicts --------------------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
```

```r
data('PimaIndiansDiabetes2')
#omit missing values
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)

#make model, probs, predictions
model <- glm(diabetes~., data=PimaIndiansDiabetes2, family=binomial)
probabilities <- predict(model, PimaIndiansDiabetes2, type='response')
pred <- ifelse(probabilities>.5, 'pos', 'neg')

#Linearity of logit: method 1
Linear <- glm(diabetes~age*log(age), family=binomial, data=PimaIndiansDiabetes2)
summary(Linear)
```

```
##
## Call:
## glm(formula = diabetes ~ age * log(age), family = binomial, data = PimaIndiansDiabetes2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4025  -0.8260  -0.4981   1.0031   2.1963
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -36.52877   32.83654  -1.112    0.266
## age           -0.77789    3.05856  -0.254    0.799
## log(age)      14.57117   20.39801   0.714    0.475
## age:log(age)   0.09673    0.54073   0.179    0.858
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 498.10  on 391  degrees of freedom
## Residual deviance: 432.04  on 388  degrees of freedom
## AIC: 440.04
##
## Number of Fisher Scoring iterations: 4
```

```
#The interaction is nonsignificant, so the assumption is passed for age)
#This should be repeated for all numeric variables

#Linearity of logit: method 2 (plot all variables at once)
numericalData <- select_if(PimaIndiansDiabetes2, is.numeric)
predictors <- colnames(numericalData) #pull column names for numeric data
numericalData <- numericalData %>% mutate(logit=log(probabilities/(1-probabilities))) %>%
  gather(key='predictors', value='predictor.value', -logit)
ggplot(numericalData, aes(logit, predictor.value)) + geom_point(size=.5) +
  geom_smooth() + facet_wrap(~predictors, scale='free_y')
```

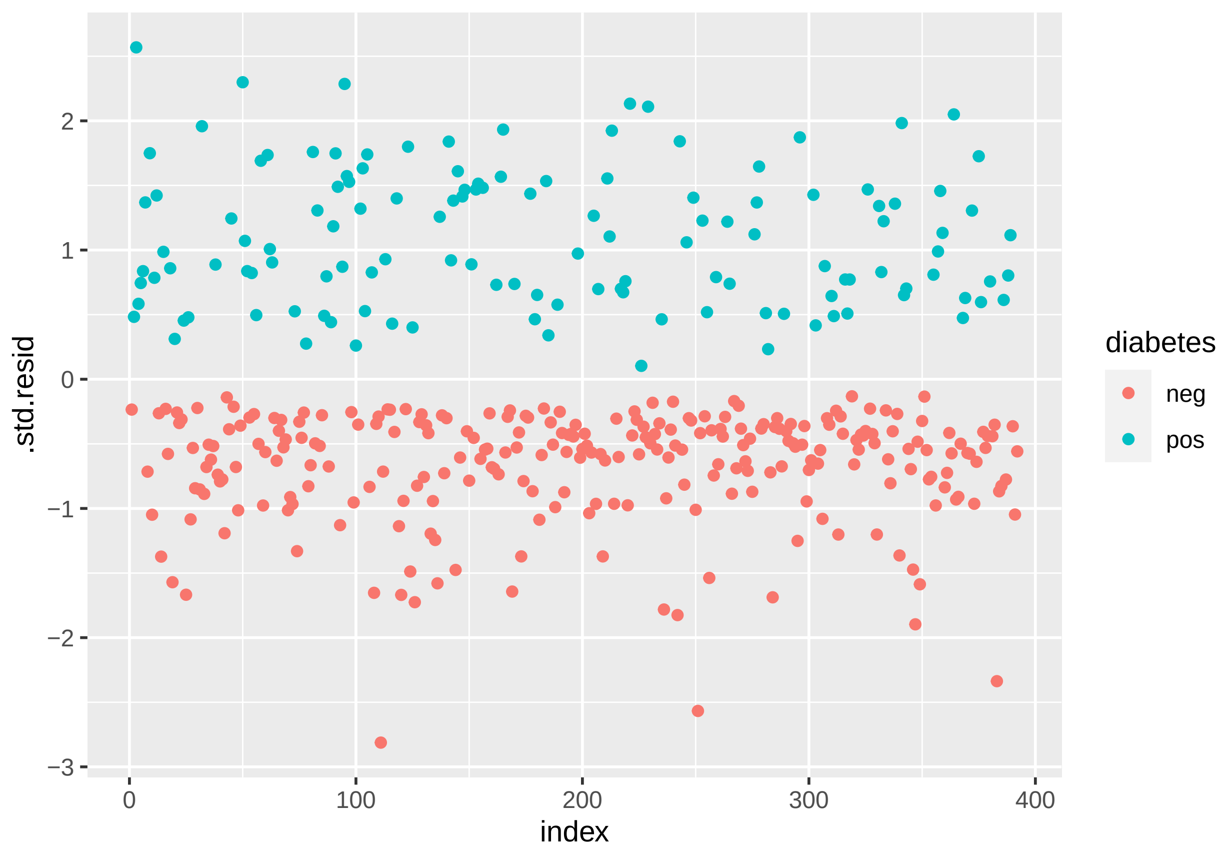
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



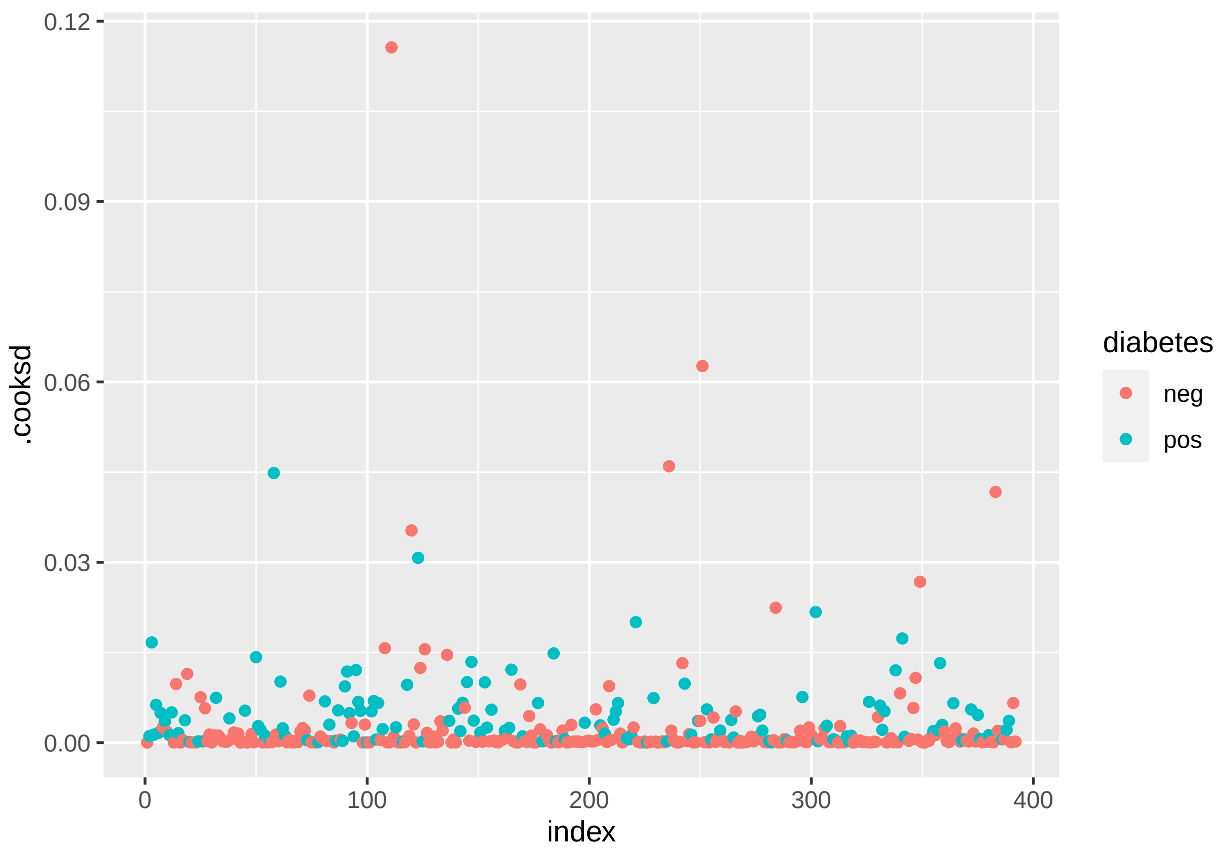
```
#All of the plots should be more or less linear

#Multicollinearity
car::vif(model)
```

```
## pregnant  glucose pressure  triceps  insulin     mass pedigree      age
## 1.892387 1.378937 1.191287 1.638865 1.384038 1.832416 1.031715 1.974053
```

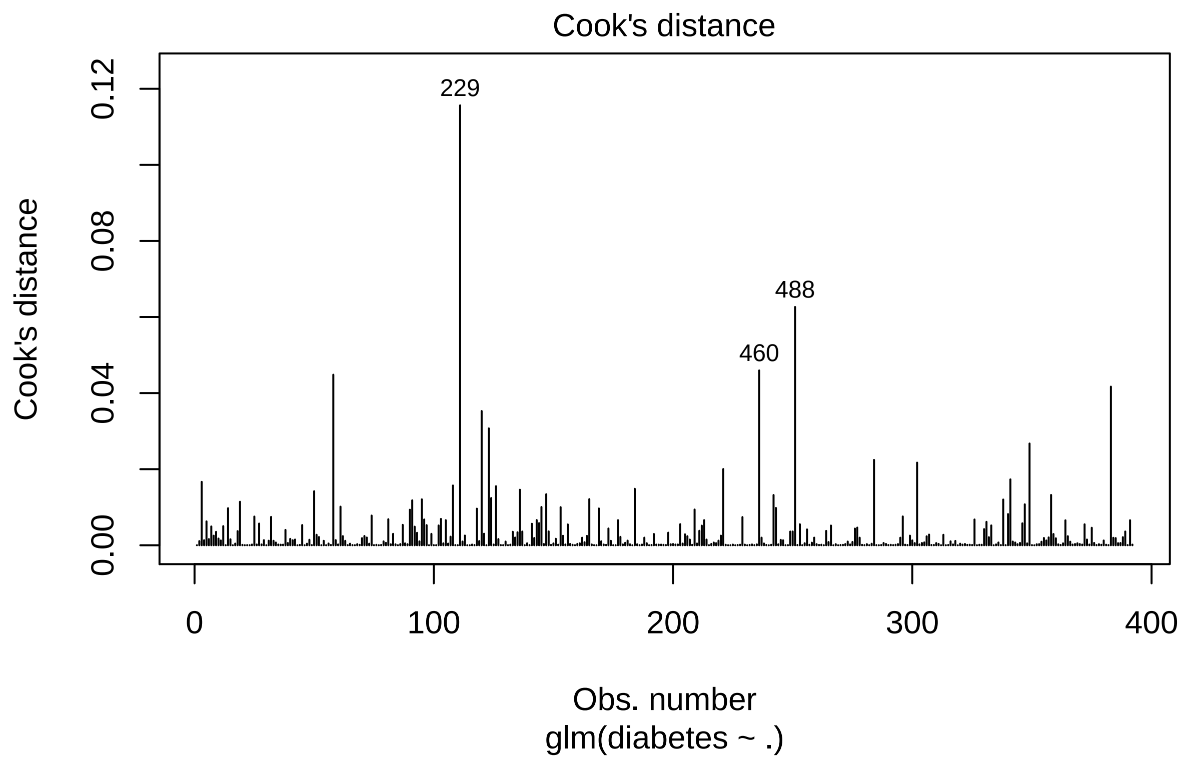
```
#Influential Outliers
modelResults <- broom::augment(model) %>% mutate(index=1:n())
ggplot(modelResults, aes(index, .std.resid)) + geom_point(aes(color=diabetes))
```



```r
ggplot(modelResults, aes(index, .cooksd)) + geom_point(aes(color=diabetes))
```



```r
plot(model, which=c(4))
```



```
#Points with a high Cook's distance are suspect
```

Linear Discriminant Analysis (LDA)

General Description

Linear discriminant analysis is a form of classification. The distribution of each prediction class X is modeled separately. Bayes is then used to determine the probability of Y (Pr(Y|X)). Each centroid and its distribution is then used to produce straight boundary lines to separate each class.

When To Use It

LDA is most effective when the classes are well separated and each predictor follows a normal distribution. Unlike logistic regression, LDA works well with small datasets and datasets with more than two response classes. However, it is only effective if each outcome group has more observations than predictor variables. The independent variables may be continuous or categorical; the dependent variable is categorical.

Modeling Functions

Like other classification models, LDA is generally run on a training dataset and tested on a test set. Data should be centered and scaled before applying LDA.

To center and scale: preprocessing<-train %>% preProcess(method=c('center','scale')) traintransformed<-preprocessing %>% predict(train) testtransformed<-preprocessing %>% predict(test)

To create the model: model <- lda(dependent~independent, data=your_data)

To make predictions: preds <- predict(model) preds$class

Modeling Function Example

```r
library(rattle)
library(MASS) #lda function is in MASS
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```r
data(wine)
attach(wine)

divideData<-Type %>% createDataPartition(p=.8,list = F)
train<-wine[divideData,]
test<-wine[-divideData,]

preprocessing<-train %>% preProcess(method=c('center','scale'))
traintransformed<-preprocessing %>% predict(train)
testtransformed<-preprocessing %>% predict(test)

model<-lda(Type~.,data=traintransformed)

#make predictions on test group
prediction<- model %>% predict(testtransformed)

#calculate the accuracy rate
mean(prediction$class == testtransformed$Type)
```

```
## [1] 1
```

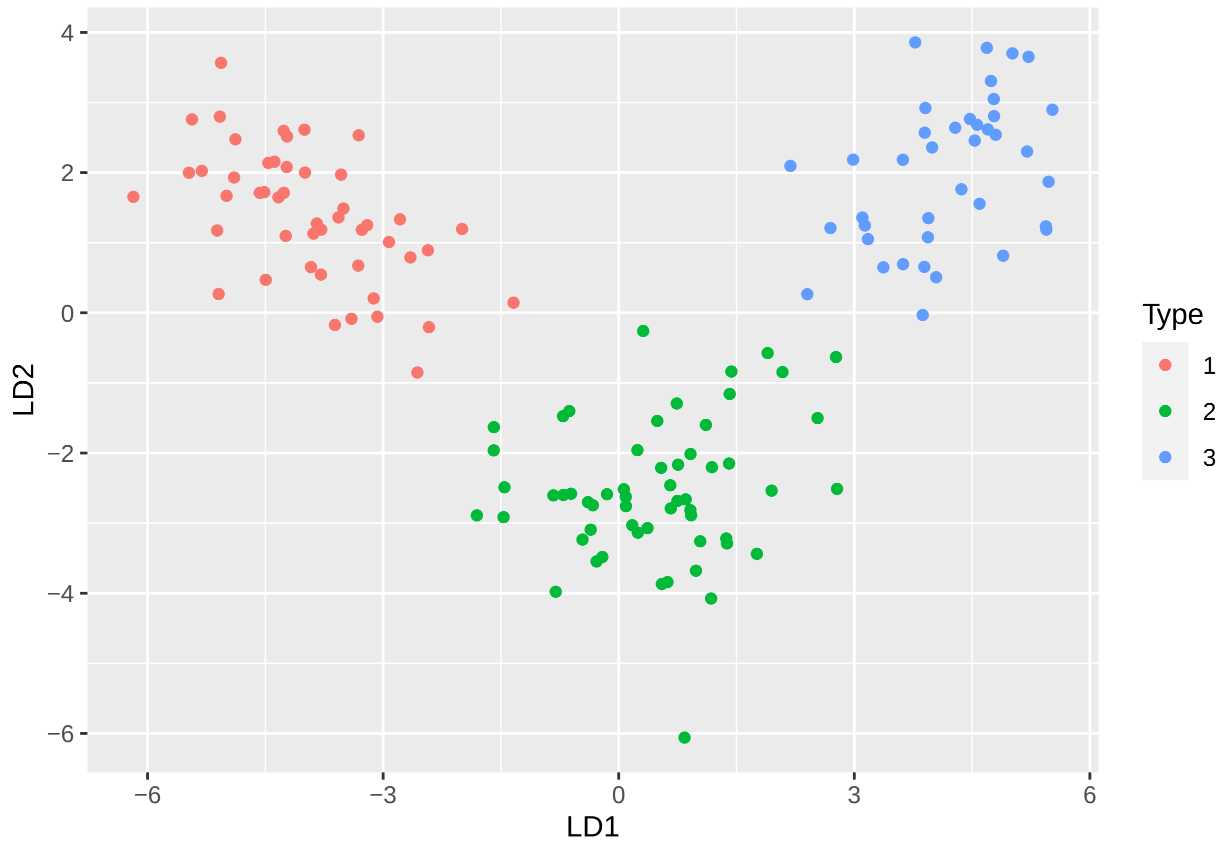
```r
#calculate the error rate
mean(prediction$class != testtransformed$Type)
```

```
## [1] 0
```

```r
# make prediction table
table(prediction$class, testtransformed$Type)
```

```
##
##       1  2  3
##   1 11  0  0
##   2  0 14  0
##   3  0  0  9
```

```r
#plot the data from the model
ldaforgraph<-cbind(traintransformed,predict(model)$x)
ggplot(ldaforgraph, aes(LD1,LD2)) +geom_point(aes(color=Type))
```



Assumptions and Tests

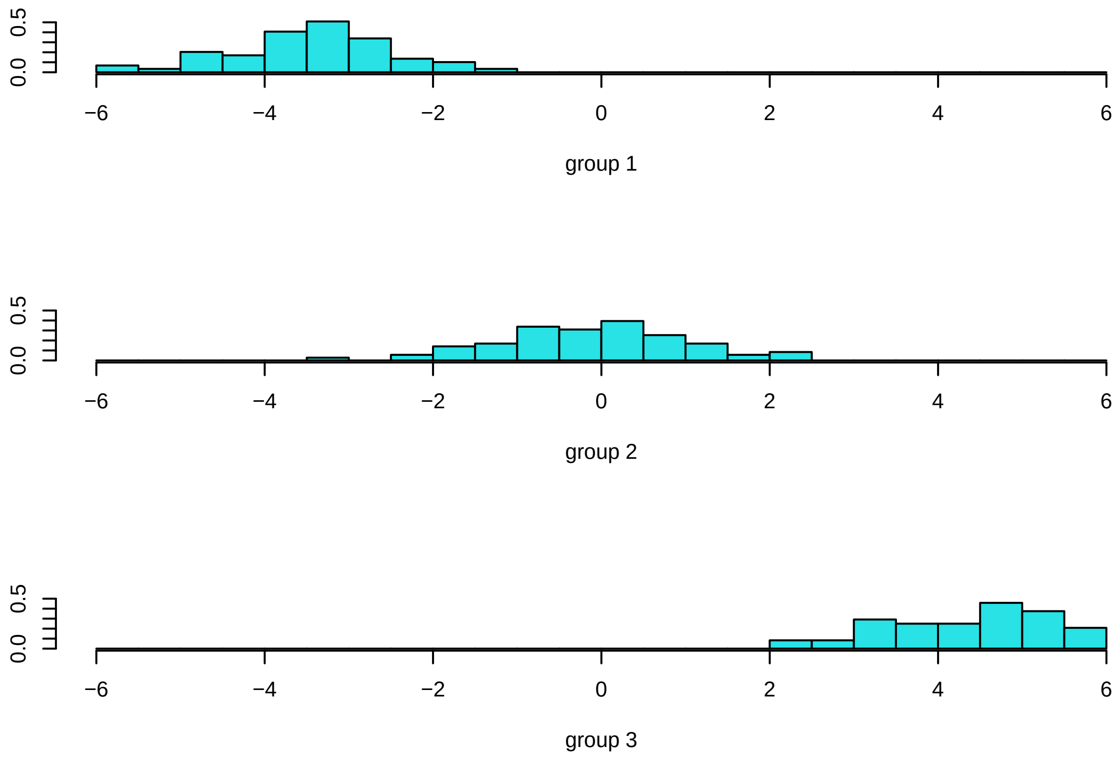
The assumptions for LDA are: 1. No strongly influential outliers 2. Multivariate normality 3. Lack of multicollinearity 4. Homoscedasticity 5. Independence of observations (no repeated observations) 6. Normality of LD functions

Refer to the multiple linear regression section for detailed explanations and examples of these assumptions. For LDA, these assumptions may be violated; it is most important to consider the overall error rate of the model.

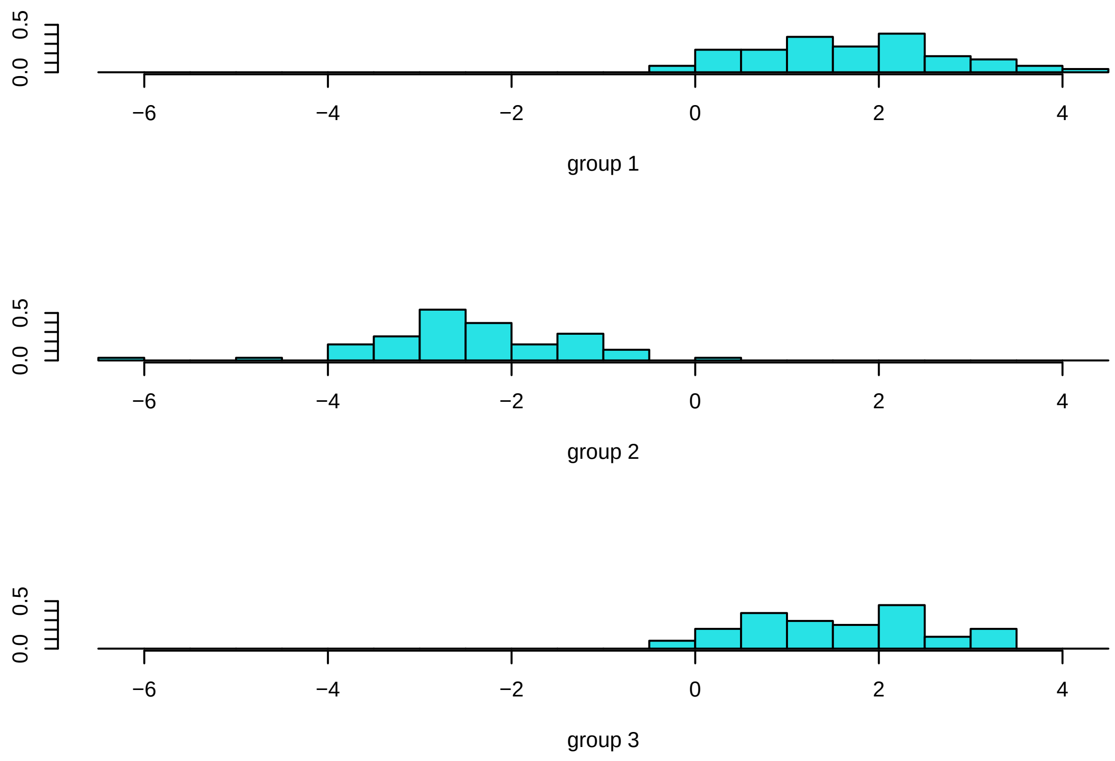
Assumptions and Tests Example

Most of the assumptions are summarized in the multiple linear regression section. One LDA-specific assumption is that the linear discriminant functions (LD1, LD2, etc) should be normal.

```
wine_lda<-lda(Type~.,data=wine)
wine_lda_values<-predict(wine_lda)
ldahist(data=wine_lda_values$x[,1], g=wine$Type)
```



```
ldahist(data=wine_lda_values$x[,2], g=wine$Type)
```



Quadratic Discriminant Analysis (QDA)

General Description

QDA is essentially the same as LDA except that the lines it uses to separate groups are not linear. This can help improve the model accuracy over LDA, but it also requires more data, and can potentially lead to over-fitting. It also removes some of the plotting and visualization potential that is seen in LDA.

When To Use It

When performing discriminant analysis, it is common to create an LDA model, a QDA model, and a KNN model. The model with the highest accuracy rate is selected. Just like for LDA, the data should be centered and scaled before applying the model.

Modeling Functions

The functions for centering and scaling and making predictions are the same as for LDA. To make the model: model<-qda(Type~.,data=traintransformed)

Modeling Function Examples

```
data(wine)
attach(wine)
```

```
## The following objects are masked from wine (pos = 3):
##
```

```
##     Alcalinity, Alcohol, Ash, Color, Dilution, Flavanoids, Hue,
##     Magnesium, Malic, Nonflavanoids, Phenols, Proanthocyanins, Proline,
##     Type
```

```
divideData<-Type %>% createDataPartition(p=.8,list = F)
train<-wine[divideData,]
test<-wine[-divideData,]

preprocessing<-train %>% preProcess(method=c('center','scale'))
traintransformed<-preprocessing %>% predict(train)
testtransformed<-preprocessing %>% predict(test)

model<-qda(Type~.,data=traintransformed)

#make predictions on test group
prediction<- model %>% predict(testtransformed)

#calculate the accuracy rate
mean(prediction$class == testtransformed$Type)
```

```
## [1] 1
```

```
#calculate the error rate
mean(prediction$class != testtransformed$Type)
```

```
## [1] 0
```

```
# make prediction table
table(prediction$class, testtransformed$Type)
```

```
##
##       1  2  3
##   1 11  0  0
##   2  0 14  0
##   3  0  0  9
```

Assumptions and Tests

The assumptions for QDA are: 1. No strongly influential outliers 2. Multivariate normality 3. Lack of multicollinearity 4. Homoscedasticity 5. Independence of observations (no repeated observations)

Refer to the multiple linear regression section for detailed explanations and examples of these assumptions.

K Nearest Neighbors

General Description

KNN uses the Bayes classifier to predict the class of a given observation given the class of its nearest neighbors. The number of neighbors is given by k; by default, R tests several different values of k and returns the model with the highest accuracy. KNN is an example of a nonparametric technique, since it makes no assumptions about the shape of the model.

When To Use It

Like LDA and QDA, KNN works on datasets that have some division between categories. However, KNN performs better when the boundaries between the categories are jagged or irregular. Generally, it is common practice to create an LDA model, a QDA model, and a KNN model for the same dataset and choose the model with the highest accuracy rate.

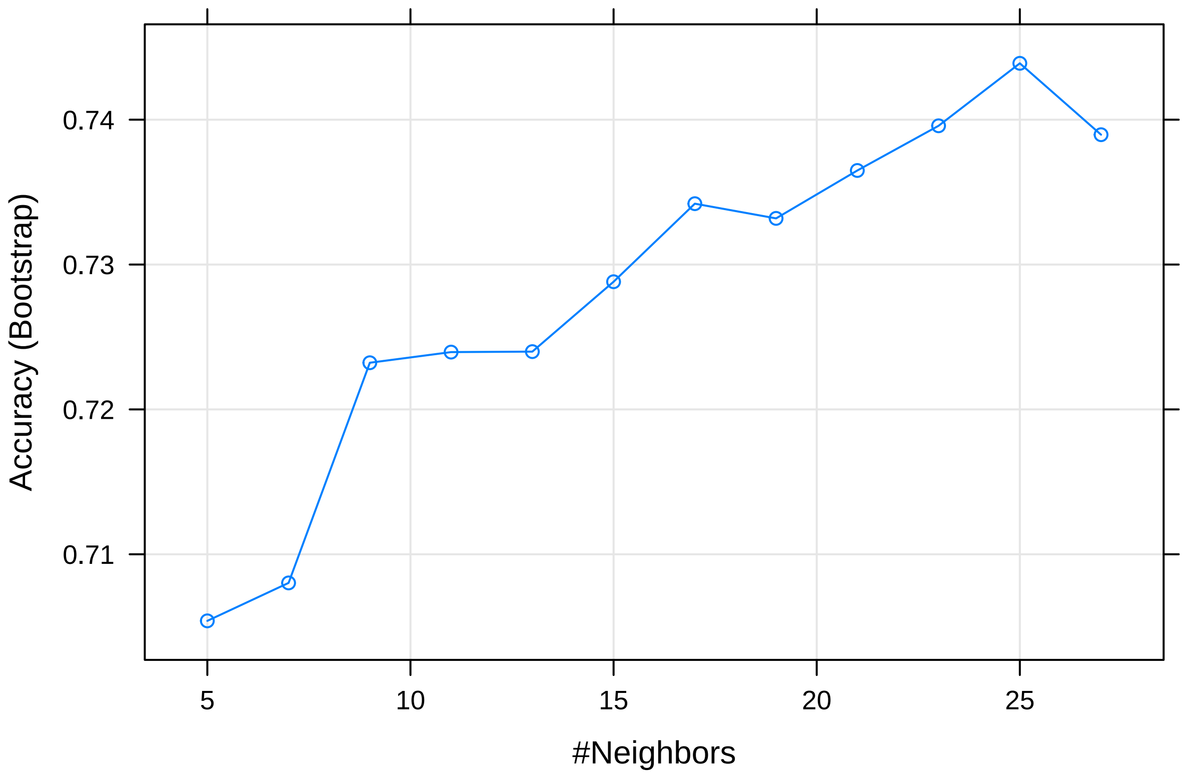
Modeling Functions

Split the data into testing and training sets as described in previous sections. To make the model (trying N different values of k): model<-train(dependent~independent,data=train,method='knn', tuneLength=N, preProcess=c('center','scale'))

To make predictions: preds<-predict(knnfit,newdata=test)

Modeling Function Examples

```
data("PimaIndiansDiabetes2")
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
divideData<-createDataPartition(PimaIndiansDiabetes2$diabetes, p=.8,list=F)
train<-PimaIndiansDiabetes2[divideData,]
test<-PimaIndiansDiabetes2[-divideData,]

### Fit the model with training data
knnfit<-train(diabetes~.,data=train,method='knn',preProcess=c('center','scale'),
              tuneLength=12)
plot(knnfit)
```



```
knnfit$bestTune #optimal value of k
```

```
##     k
## 11 25
```

```r
### make prediction on the test dataset
knnclass<-predict(knnfit,newdata=test)
head(knnclass)
```

```
## [1] neg pos pos neg neg neg
## Levels: neg pos
```

```r
### confusion matrix
confusionMatrix(knnclass,test$diabetes)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction neg pos
##        neg  51  14
##        pos   1  12
##
##                Accuracy : 0.8077
##                  95% CI : (0.7027, 0.8882)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 0.004395
##
##                   Kappa : 0.5055
##
##  Mcnemar's Test P-Value : 0.001946
##
##             Sensitivity : 0.9808
##             Specificity : 0.4615
##          Pos Pred Value : 0.7846
##          Neg Pred Value : 0.9231
##              Prevalence : 0.6667
##          Detection Rate : 0.6538
##    Detection Prevalence : 0.8333
##       Balanced Accuracy : 0.7212
##
##        'Positive' Class : neg
##
```

```r
### calculate accuracy rate
mean(knnclass==test$diabetes)
```

```
## [1] 0.8076923
```

Assumptions and Tests

There are no assumptions or tests necessary for KNN.

Part 3: Validation/Resampling

Resampling is used to refit a model to samples from the training set in order to more accurately assess the test set error. The resampling method is used together with one of the prediction or classification techniques. Generally, the technique that yields the best statistics (highest R^2, lowest MAE/MSE/RMSE) is selected.

Validation Set Approach

General Summary

In the validation set approach, the dataset is divided into testing and training groups (just like for classification models). The model is created with the training group and predictions are made using the testing group. There are two major drawbacks to this approach; first, the estimates can be highly variable, depending on what points are included in the training set. Second, models perform worse when trained on fewer observations, so excluding the testing set could cause a systematic misestimation of error rates.

Validation Set Example

```r
library(ISLR)
library(mltools)
```

```
##
## Attaching package: 'mltools'

## The following object is masked from 'package:tidyr':
##
##     replace_na
```

```r
data(Auto)
divideData <- createDataPartition(Auto$mpg, p=0.75, list=FALSE)
train <- Auto[divideData, ]
test <- Auto[-divideData, ]
lm.fit <- lm(mpg~poly(horsepower, 2), data=train)
newpred <- predict(lm.fit, newdata=test)

mse <- mse(newpred, test$mpg)
RMSE <- rmse(newpred, test$mpg)
RSquare <- R2(newpred, test$mpg)
mae <- MAE(newpred, test$mpg)
```

Leave One Out Cross-Validation (LOOCV)

General Summary

LOOCV seeks to correct the problems introduced by the validation set approach. In LOOCV, one data point is held as the testing set, and the model is trained with all remaining points. The process is repeated so each point is held out as the validation point. Although the method reduces bias, it is costly to implement with large datasets. Importantly, it will always yield the same results.

LOOCV Example (with linear model)

```r
divideData <- createDataPartition(Auto$mpg, p=0.7, list=FALSE)
train <- Auto[divideData, ]
test <- Auto[-divideData, ]

trainControl <- trainControl(method='LOOCV')
LOOCVmodel <- train(mpg~horsepower, data=train, method='lm', trControl=trainControl)
newpred <- predict(LOOCVmodel, newdata=test)

mse <- mse(newpred, test$mpg)
RMSE <- rmse(newpred, test$mpg)
```

```
RSquare <- R2(newpred, test$mpg)
mae <- MAE(newpred, test$mpg)
```

K-Fold Cross Validation

General Summary

K-fold CV involves splitting the dataset into k number of subsets. It then sets one subset as the test set and trains the model on the others. This process is repeated k times so each subset serves as the test set. The value of k is usually set to 5 or 10. K-fold validation strikes a balance between the bias introduced by the validation set approach and the computational intensity of LOOCV.

K-Fold Example (with LDA)

```
data('iris')
divideData <- createDataPartition(iris$Species, p=0.75, list=FALSE)
train <- iris[divideData, ]
test <- iris[-divideData, ]

trainControl <- trainControl(method='cv', number=10)
ldamodel <- train(Species~., method='lda', data=train, trControl=trainControl)
newpred <- predict(ldamodel, newdata=test)
mean(newpred==test$Species) #accuracy rate
```

```
## [1] 1
```

```
confusionMatrix(newpred, test$Species) #confusion matrix
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   setosa versicolor virginica
##   setosa         12          0         0
##   versicolor      0         12         0
##   virginica       0          0        12
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9026, 1)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                 1.0000            1.0000           1.0000
## Specificity                 1.0000            1.0000           1.0000
## Pos Pred Value              1.0000            1.0000           1.0000
## Neg Pred Value              1.0000            1.0000           1.0000
## Prevalence                0.3333            0.3333           0.3333
## Detection Rate            0.3333            0.3333           0.3333
## Detection Prevalence      0.3333            0.3333           0.3333
## Balanced Accuracy         1.0000            1.0000           1.0000
```

```
table(newpred, test$Species)
```

```
##
## newpred      setosa versicolor virginica
##   setosa         12          0         0
##   versicolor      0         12         0
##   virginica       0          0        12
```