Post02 - Classification Methods in R

JackMoorer

Introduction

Classification is an important aspect of statistical learning. Classification refers to supervised statistical learning methods where response values are discrete. In order to distinguish between classification and linear regression I will provide an example for the reader.

Say I want to predict the amount of sales based on advertising budget, the model I would use would be a regression model, because sales, our current response variable, is continuous. In the simple linear regression cause we can fit a model like:

\[y = \hat{B_0} + \hat{B_1}x \] We have seen something like this in class and homework. When our response variable, y, is discrete, however, classic regression is not ideal. For example, say we want to fit a model to determine based on plant width what species, out of three possible species, a certain plant is. In this case regression is not ideal. We want to fit a model to predict either plant 1, 2, or 3. I will later show why regression is not ideal.

This post will go over several classification methods in r – Logistic Regression, Discriminant Analysis, and K-nearest neighbors. I will also provide information on strengths of each method and information and when to use a specific method.

Methods

Logistic Regression

In this section I am going to be using the Default data set from the package ISLR. ISLR is the package for the book Introduction to Statistical Learning with Application in R. In order to run my code please do the following:

```
#install the package
#NOTE: Please install this package by running the following line on your console!!!!!
install.packages("ISLR")
```

```
#load the package
library(ISLR)
```

```
#Let's look at the head of default
head(Default)
```

```
##   default student   balance    income
## 1      No      No  729.5265 44361.625
## 2      No     Yes  817.1804 12106.135
## 3      No      No 1073.5492 31767.139
## 4      No      No  529.2506 35704.494
## 5      No      No  785.6559 38463.496
## 6      No     Yes  919.5885  7491.559
```

As we can see the default data set has four columns. Two are categorical variables, Default – yes or no – and Student – yes or no. These are discrete variables. The data set also has 2 continuous variables – balance and income.

Let's start by looking at linear regression. Let me say I think there is a relationship between balance and income. Let me set balance as my response variable Y, and income as my predictor variable X. We can fit a regression model using the function lm.

```
#fit regression model
regression <- lm(balance ~ income, data = Default)
#print the summary of our regression
summary(regression)
```

```
##
## Call:
## lm(formula = balance ~ income, data = Default)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -968.98 -349.08   -8.62  323.27 1762.96
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.020e+03  1.293e+01   78.91   <2e-16 ***
## income      -5.522e-03  3.585e-04  -15.40   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 478.1 on 9998 degrees of freedom
## Multiple R-squared:  0.02318,    Adjusted R-squared:  0.02308
## F-statistic: 237.2 on 1 and 9998 DF,  p-value: < 2.2e-16
```

```
#install the package
#NOTE: Please install this package by running the following line on your console!!!!!
install.packages("ggplot2")
```

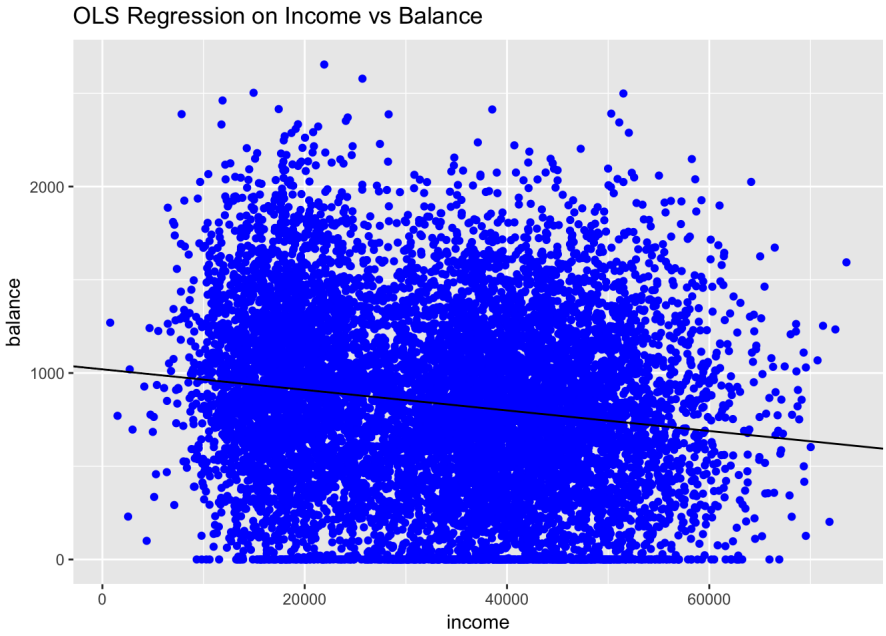
```r
#load the package
library(ggplot2)
```

```r
#look at the coefficients of our model
coefficients <- regression$coefficients
coefficients
```

```
##   (Intercept)        income
##  1.020449e+03 -5.521812e-03
```

We can now look at our linear model using geom_abline:

```r
#initialize ggplot object
g <- ggplot(data = Default, aes(x = income, y = balance))
#add scatterplot
g <- g + geom_point(col = "blue")
#add regression line
g <- g + geom_abline(intercept = coefficients[[1]], slope = coefficients[[2]])
#add title
g <- g + ggtitle("OLS Regression on Income vs Balance")
#show graph
g
```



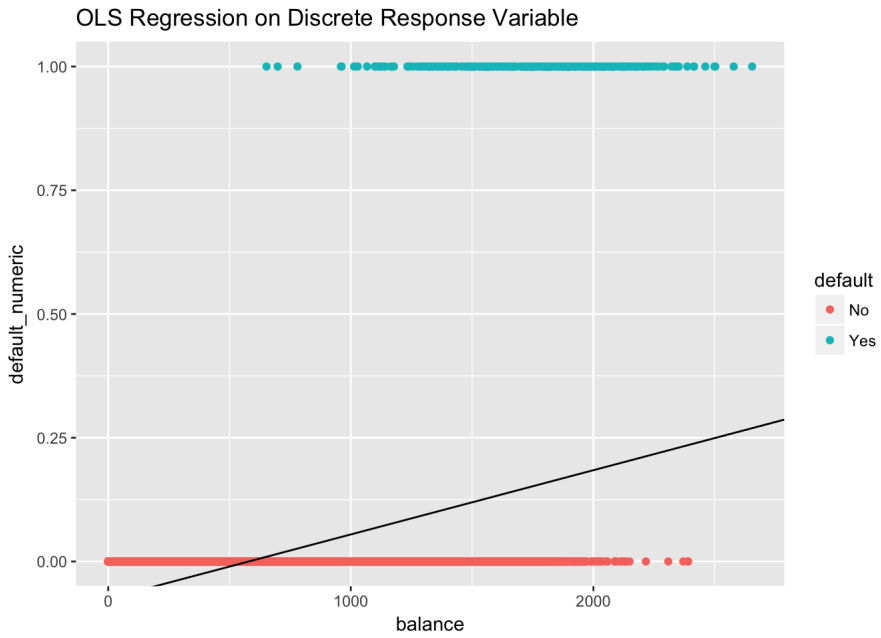
We can see there is not much of a relationship in this case between income and balance, but r still made a regression line predicting the balance of a person based on their income.

Moving on to classification, what if we now wanted to find the relationship between default rate and balance. Here, our response variable, y, is the variable Default, which is a discrete variable. Let's look at what happens when we fit a linear model.

```r
#First we need to change Default to a numeric number
default_numeric <- rep(0, nrow(Default))
default_numeric[Default$default == 'Yes'] <- 1
Default$default_numeric <- default_numeric
#fit a linear model
ols_reg <- lm(default_numeric ~ balance, data = Default)
#find the coefficients
coef <- ols_reg$coefficients
```

Let's plot our linear model, sepearting the default status with blue as default = yes and red as default = no:

```r
#intialize ggplot
g <- ggplot(data = Default, aes(x = balance, y = default_numeric, color = default))
#make scatterplot
g <- g + geom_point()
#add regression line
g <- g + geom_abline(intercept = coef[[1]], slope = coef[[2]])
#add title
g <- g + ggtitle("OLS Regression on Discrete Response Variable")
#look at plot
g
```



There are two issues with this plot. For one, aesthetically, this doesn't look great. The more serious issue is how can we interpret and classify our result? One way to do this is to look at the estimated values on our regression line as probabilities. We can classify as Default = Yes if the value above 0.5, so there is greater than 50% probability that they will default. There are two issues with using regression in this case. For one, regression isn't going to do very well. The other is that regression is continuous, so we will have estimated values outside the range of [0, 1]. If we want just probabilities, we want the lowest value to be 0, and the greatest value to be 1.

This is where logistic regression comes in to play. Logistic regression uses a sigmoid function to express the regression model as probabilities.

This is the function we use:

\[p(x) = \frac{e^{\hat{B_0} + \hat{B_1}x}}{1 + e^{\hat{B_0} + \hat{B_1}x}} \] And we classify X as in category 1 for p(x) > 0.5, and p(x) in category 0 for p(x) < 0.5.

Let's look at our logistic regression model with default and balance.

```
#make logistic model
log <- glm(default ~ balance, data = Default, family=binomial(link="logit"))
#let's look at the coefficients
summary(log)$coefficients
```

```
##                 Estimate   Std. Error    z value      Pr(>|z|)
## (Intercept) -10.651330614 0.3611573721 -29.49221 3.623124e-191
## balance       0.005498917 0.0002203702  24.95309 1.976602e-137
```

Now we can classify values based on an inputed balance using predict() with type = "response".

Let's predict a series of values, 100, 200, 300, ..., 1900, 2000

```
#make data frame of balance values to predict
bal <- data.frame(balance = seq(100, 2000, by = 100))
#predict default status
predict(log, bal, type = "response")
```

```
##            1            2            3            4            5
## 4.101880e-05 7.108613e-05 1.231905e-04 2.134779e-04 3.699132e-04
##            6            7            8            9           10
## 6.409100e-04 1.110217e-03 1.922514e-03 3.327154e-03 5.752145e-03
##           11           12           13           14           15
## 9.926984e-03 1.707982e-02 2.923441e-02 4.960213e-02 8.294762e-02
##           16           17           18           19           20
## 1.355136e-01 2.136317e-01 3.201070e-01 4.493274e-01 5.857694e-01
```

These values are probabilities. We can predict whether a person will default or not based on our classification rule p(x) > 0.5. In the example above only the person with a balance of 2000 we would predict to default.

Let's see how well our model does on its own training set:

```
#prediction probabilities
preds <- predict(log, data.frame(balance = Default$balance), type = "response")
#intialize empty classification vector
classification <- rep(0, nrow(Default))
#classify default status
classification[preds > 0.5] <- "Yes"
classification[preds <= 0.5] <- "No"
```

I am going to look at the error rate of the model:

```
#real is the true default status
real <- Default$default
#see error rate
(1/nrow(Default))*sum(real != classification)
```

```
## [1] 0.0275
```

As you can see our model only incorrectly classifies default status 2.75% of the time, not bad! However, whenever we look at the error rate of a training set, we are getting a dishonest evaluation of our model. To truly test the error rate we should test the model on data it has never seen before, however, I will skip this idea for now.

Estimating coefficients

You might be wondering how the coefficients are estimated in logistic regression. Let's look our logistic regression model on a new data set, Smarket, also from the ISLR library:

```
summary(Smarket)
```

```
##       Year           Lag1                Lag2          
##  Min.   :2001   Min.   :-4.922000   Min.   :-4.922000  
##  1st Qu.:2002   1st Qu.:-0.639500   1st Qu.:-0.639500  
##  Median :2003   Median : 0.039000   Median : 0.039000  
##  Mean   :2003   Mean   : 0.003834   Mean   : 0.003919  
##  3rd Qu.:2004   3rd Qu.: 0.596750   3rd Qu.: 0.596750  
##  Max.   :2005   Max.   : 5.733000   Max.   : 5.733000  
##       Lag3                Lag4                Lag5          
##  Min.   :-4.922000   Min.   :-4.922000   Min.   :-4.92200  
##  1st Qu.:-0.640000   1st Qu.:-0.640000   1st Qu.:-0.64000  
##  Median : 0.038500   Median : 0.038500   Median : 0.03850  
##  Mean   : 0.001716   Mean   : 0.001636   Mean   : 0.00561  
##  3rd Qu.: 0.596750   3rd Qu.: 0.596750   3rd Qu.: 0.59700  
##  Max.   : 5.733000   Max.   : 5.733000   Max.   : 5.73300  
##      Volume           Today           Direction 
##  Min.   :0.3561   Min.   :-4.922000   Down:602  
##  1st Qu.:1.2574   1st Qu.:-0.639500   Up  :648  
##  Median :1.4229   Median : 0.038500             
##  Mean   :1.4783   Mean   : 0.003138             
##  3rd Qu.:1.6417   3rd Qu.: 0.596750             
##  Max.   :3.1525   Max.   : 5.733000
```

I will fit a model with Direction as the response, and Lag1, Lag2, Lag3, Lag4, Lag5, and Volume as predictors.

```
#fit our model
log_fit <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
               data = Smarket, family = binomial)
#look at summary
summary(log_fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = Smarket)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.446  -1.203   1.065   1.145   1.326
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.126000   0.240736  -0.523    0.601
## Lag1        -0.073074   0.050167  -1.457    0.145
## Lag2        -0.042301   0.050086  -0.845    0.398
## Lag3         0.011085   0.049939   0.222    0.824
## Lag4         0.009359   0.049974   0.187    0.851
## Lag5         0.010313   0.049511   0.208    0.835
## Volume       0.135441   0.158360   0.855    0.392
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1731.2  on 1249  degrees of freedom
## Residual deviance: 1727.6  on 1243  degrees of freedom
## AIC: 1741.6
##
## Number of Fisher Scoring iterations: 3
```

Let's look at just the coefficients

```
#coefficients of our fit
log_fit$coefficients
```

```
##  (Intercept)         Lag1         Lag2         Lag3         Lag4
## -0.126000257 -0.073073746 -0.042301344  0.011085108  0.009358938
##         Lag5       Volume
##  0.010313068  0.135440659
```

Unlike linear regression, there is no closed form solution to finding the estimated coefficients. Instead we use something classed the Newton-Raphson algorithm.

Here is the pseudo-code for Newton-Raphson: \[y = vector \ of \ response, \ X \ is \ the \ matrix \ of \ predictors \] \[set \ b_{old} = 0, \ b_{old} \ is \ a \ vector \ of \ predictors \] \[p[x_i] = \frac{e^{x_ib_{old}}}{1 + e^{x_ib_{old}}}, \ for \ each \ row \] \[\hat{X} \ is \ formed \ by \ multiplying \ the \ ith \ row \ of \ X \ by \ p(x_i)(1 -p(x_i))\] \[b_{new} = b_{old} + (X^T\hat{X})^{-1}X^T(y - p) \] If bnew is close in value to bold then the algorithm has converged and we stop. Here is my own r-code for the algorithm:

```
#get number of rows
n <- nrow(Smarket)
#set up matrix of predictors
X <- model.matrix(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
                  data = Smarket)
# set up b_old
b_old <- rep(0, ncol(X))
#set up p
p <- rep(0, n)
#intalize X_hat
X_hat <- X
#intialize y, make it numeric with 1 equal to up and 0 equal to down
smarket_numeric <- rep(0, nrow(Smarket))
smarket_numeric[Smarket$Direction == 'Up'] <- 1
Smarket$Direction <- smarket_numeric
y <- Smarket$Direction
repeat {
  #loop all rows
  for (i in 1:n) {
    #get ith row
    xi <- X[i, ]
    #get p[xi]
    p_xi <- (exp(t(xi)%*%b_old))/(1 + exp(t(xi)%*%b_old))
    #updata p vector
    p[i] <- p_xi
    #update X_hat
    X_hat[i,] <- xi%*%p_xi
  }
  #make b_new
  b_new <- b_old + solve(t(X)%*%X_hat)%*%t(X)%*%(y - p)
  #check if less than tolerance
  if (sum(abs(b_new - b_old)) < (10^-10)) {
    break
  }
  else {
    b_old <- b_new
  }
}
#print new coefficients
b_new
```

```
##                    [,1]
## (Intercept) -0.126000259
## Lag1        -0.073073747
## Lag2        -0.042301345
## Lag3         0.011085108
## Lag4         0.009358938
## Lag5         0.010313069
## Volume       0.135440661
```

As you can see we got the same estimated coefficients (approximately) as our model using glm().

Discriminant Analysis

Linear Discriminant Analysis

Before I begin with Linear Discriminant Analysis (LDA) please download the following package:

```
#install the package
#NOTE: Please install this package by running the following line on your console!!!!!
install.packages("MASS")
```

```
#load the package
library(MASS)
```

I am going to use the data set iris:

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

I am going to use LDA to create a model to classify the type of species based on the four predictors Sepal Length, Sepal Width, Petal Length, and Petal Width.

First I am going to split the data into training and test sets. The reason I am doing this is that, when one tries to evaluate how well a statistical learning model performs, it is important to test the data on data the model has never seen before. Then using the predict() function, I am going to predict what species the unseen data in the test data set is based on the LDA model.

In order to create a training set and test set I am going to generate a random sample of 80 percent of the data, using it as the training data, and then use the remaining 20% as the test data.

```
#seed set
set.seed(100)
#generate random sample index
smp_size <- floor(0.8*nrow(iris))
train_index <- sample(1:nrow(iris), size = smp_size)
#seperate into train and test data sets
train = iris[train_index, ]
test = iris[-train_index, ]
#built the lda model
lda <- lda(Species ~ ., data=train)
#predict the test set using the lda model
predict <- predict(lda, test[,-5])
predict
```

```
## $class
##  [1] setosa     setosa     setosa     setosa     setosa     setosa
##  [7] setosa     setosa     setosa     setosa     setosa     setosa
## [13] setosa     setosa     versicolor versicolor versicolor versicolor
## [19] versicolor versicolor versicolor versicolor versicolor versicolor
## [25] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
##
## $posterior
##           setosa   versicolor    virginica
## 5   1.000000e+00 4.863077e-22 1.442075e-42
## 6   1.000000e+00 1.748620e-20 9.994855e-40
## 8   1.000000e+00 9.077931e-20 1.245967e-39
## 10  1.000000e+00 1.887531e-18 2.550449e-38
## 11  1.000000e+00 6.177436e-23 1.003183e-43
## 14  1.000000e+00 1.138876e-19 5.014007e-40
## 18  1.000000e+00 1.371224e-20 2.056697e-40
## 23  1.000000e+00 1.697921e-24 6.896908e-46
## 29  1.000000e+00 3.352843e-21 1.393335e-41
## 30  1.000000e+00 1.115290e-16 1.423617e-35
## 36  1.000000e+00 4.796989e-21 1.646399e-41
## 41  1.000000e+00 3.333190e-21 3.041495e-41
## 44  1.000000e+00 1.891339e-15 1.201299e-32
## 49  1.000000e+00 1.157666e-22 2.372299e-43
## 54  1.894494e-21 9.997438e-01 2.561882e-04
## 56  7.043803e-22 9.991701e-01 8.299307e-04
## 62  2.380896e-19 9.995220e-01 4.780023e-04
## 65  1.780860e-13 9.999990e-01 9.873640e-07
## 68  1.174490e-15 9.999993e-01 6.810534e-07
## 73  5.107784e-28 8.118566e-01 1.881434e-01
## 74  1.570450e-21 9.996702e-01 3.297601e-04
## 83  6.053446e-16 9.999971e-01 2.889927e-06
## 87  6.534731e-21 9.980917e-01 1.908284e-03
## 94  1.643359e-13 9.999999e-01 6.363416e-08
## 107 4.219215e-31 1.442540e-01 8.557460e-01
## 122 1.279629e-35 1.978835e-03 9.980212e-01
## 128 8.570501e-29 2.053579e-01 7.946421e-01
## 130 3.955751e-32 7.898182e-02 9.210182e-01
## 139 5.118786e-28 2.981423e-01 7.018577e-01
## 144 1.084015e-44 1.399682e-06 9.999986e-01
##
## $x
##            LD1         LD2
## 5    8.5613964  0.53433688
## 6    8.0848483  1.45590399
## 8    8.0468337  0.04259112
## 10   7.8070232 -0.83917396
## 11   8.7642698  0.72997555
## 14   8.1027163 -0.77201367
## 18   8.1910703  0.64388319
## 23   9.1395406  0.84866204
## 29   8.3862450  0.20233768
## 30   7.3373956 -0.53031992
## 36   8.3702487 -0.02660912
## 41   8.3356063  0.72246448
## 44   6.8588716  1.23219307
## 49   8.6993114  0.70292046
## 54  -1.6538664 -1.55563952
## 56  -1.8087511 -1.06443779
## 62  -1.3107834  0.24378274
## 65   0.1642748  0.05228924
## 68  -0.2100977 -1.63452960
## 73  -3.2800818 -1.35272502
## 74  -1.6851559 -1.44303610
## 83  -0.3566096 -0.87216502
## 87  -1.6860542  0.12509625
## 94   0.3360540 -1.77809804
## 107 -3.9170228 -0.75785895
## 122 -4.6909405  0.41478297
## 128 -3.4955792  0.38507599
## 130 -4.1009858 -0.81913726
## 139 -3.3510431  0.46365728
## 144 -6.2444987  1.28193675
```

Above you can see predict is a list with items class, posterior, and x. Before going over posterior, I will go over the results of the predicted class and x items.

Class is the predicted class of the unseen test set in iris. We can compare these predictions to the actual values to see how well our model did.

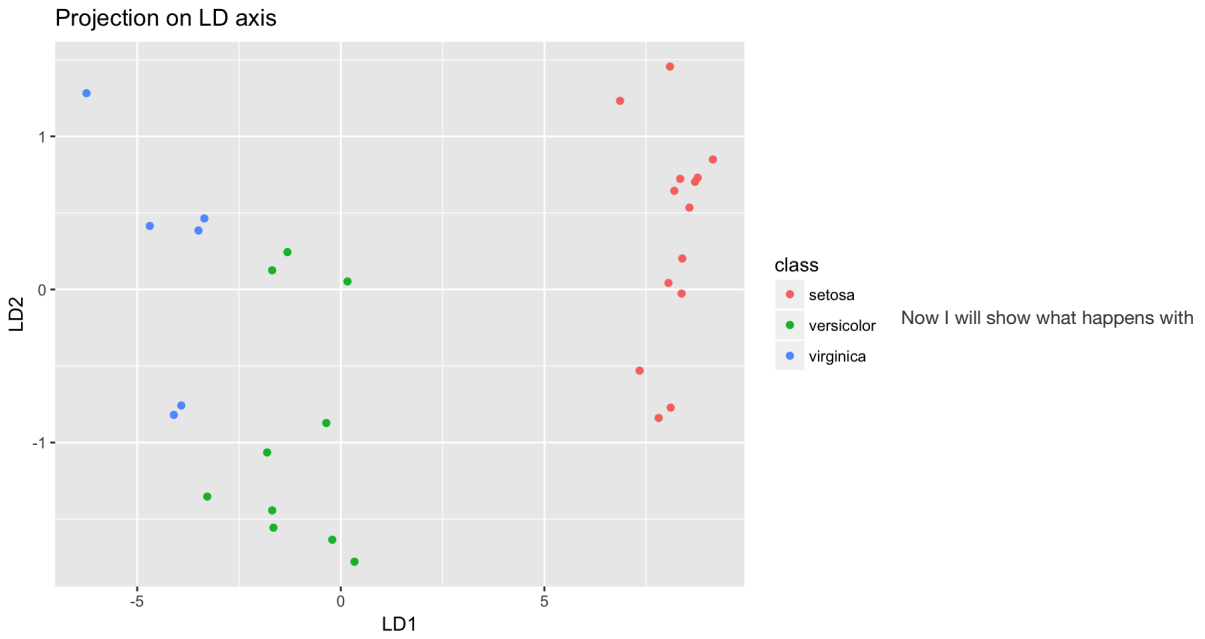
```
#get lda predictions
preds <- predict$class
#actual classes
real <- test$Species
#get error rate
err_rate <- (1/nrow(test))*sum(preds != real)
err_rate
```

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

In this case I got a 0% error rate. That is pretty luck, but doesn't really mean the lda model above is 100% correct.

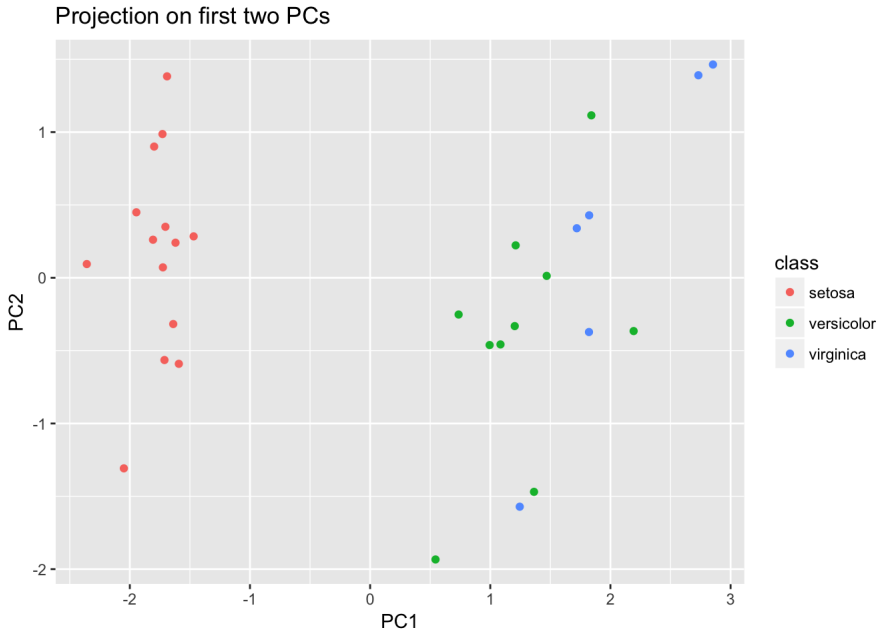
Discriminant Analysis, geometrically, is a dimension reduction technique, like PCA. The x values from predict are the vectors that span the linear discriminant axis. I am going to leave out how these vectors are computed, it is a lot of linear algebra. These vectors are the projections of the data onto the LD axis, like in PCA. I will show the difference between how the LD axis separate the data and how the Principal Axis from PCA separate the data.

```
#get data frame of LD axis
Z <- data.frame(predict$x)
#add class to data frame
Z$class <- test$Species

#plot LDA scatterplot
g <- ggplot(data = Z, aes(x = LD1, y = LD2, color = class))
#scatterplot
g <- g + geom_point()
#add title
g <- g + ggtitle("Projection on LD axis")
#print plot
g
```



Now I will show what happens with PCA:

```
#matrix of predictors
X <- test[, -5]
#run pca on predictors
pca <- prcomp(test[ ,-5], scale. = TRUE)
#get pc axis
PC <- as.data.frame(pca$x)
#set class for plotting
PC$class <- test$Species

#plot LDA scatterplot
g <- ggplot(data = PC, aes(x = PC1, y = PC2, color = class))
g <- g + geom_point()
g <- g + ggtitle("Projection on first two PCs")
g
```



The above plots show how LDA does a better job at separating species than PCA, the dimension reduction method we have seen in class. The first plot shows a clear separation between each of the classes. While the second plot on the PC axis clearly separates setosa like the first plot, the PC projections do not do well seperating versicolor and virginica.

So you can see how r lets us predict classes fairly easily using the lda() function from the MASS library, but I still haven't explained how LDA works. Previously, in discussing logistic regression I showed that logistic regression classified data using probabilities. LDA also classifies the data based on probabilities, however, it finds these probabilities in a different way. Where as logistic regression looks directly at P(Y = Class 1| X= x) by estimating the coefficients B0 and B1, LDA uses Bayes theorem to estimate the probability for each class. Look at the LaTeX below.

Bayes Thm: \[P(Y = k | X = x) = \frac{P(X = x | Y = k)*P(Y = k)}{P(X = x)} \] It can also be shown: \[P(Y = k | X = x) = \frac{P(X = x | Y = k)P(Y = k)}{\sum_{i = 1}^{K}{P(X = x | Y = i)P(Y = i)}} \] In the case of classification, say we have K classes. We look at the probability for each of each class, and classify based on the greatest probability.

LDA predicts P(Y = k) for each class by:

\[P(Y = k) = \frac{n_k}{n} \] For nk equal to the number of observations in class k, and n equal to the number of observation total in the dataset.

The way LDA uses Bayes theorem, is it assumes the distribution of P(X = x|Y = k) is approximately normal. For each k we look at the pdf of X given Y for each class:

\[f_k(x) = \frac{e^{-\frac{(x - \mu_k)^2}{2\sigma}}}{(2\pi)^{\frac{1}{2}}\sigma^{\frac{1}{2}}} \] We then classify x in class k for which pdf is the greatest.

To estimate the distribution function for X given y = k for each class, k, we need to estimate the mean of each class form the data, and the standard deviation of the data. The standard deviation is estimated as the sample variance-covariance matrix. I will leave out how to do this, but there are cool ways to do this in r.

The important thing about LDA is that each standard deviation, estimated as the sample variance-covariance matrix, is the same for each class pdf of X given Y = k. The next method I will talk about is Quadratic Discriminant Analysis (QDA), which is different in that it allows for a different estimate of the standard deviation for each class's pdf.

QDA in R

QDA is the same as LDA, except that each sigma_k, the estimated standard deviation for the distribution of X given Y = k, is different for each class.

Both LDA and QDA assume that the distribution of X given Y = k is approximately normal, and uses Bayes theorem to estimate the Probability of P(Y = k | X = x)

Here is the pdf of each X given Y = k for QDA: \[f_k(x) = \frac{e^{-\frac{(x - \mu_k)^2}{2\sigma}}}{(2\pi)^{\frac{1}{2}}\sigma_k^{\frac{1}{2}}} \] Notice the small difference.

This small difference, however, changes what is known as the decision boundary of our classification method. In LDA is can be shown that the decision boundaries for classification is linear, while the decision boundaries for QDA are quadratic, hence the name. Below is the proof on why the decision boundaries are linear in LDA and quadratic in QDA.

A Decision boundary is a point where the distribution of P(Y = k | X = x) > P(Y = j | X = x) for k not equal to l.

For LDA we have \[P(Y = k | X = x) = \frac{f_k(x)P(Y = k)}{P(X = x)} > \frac{f_j(x)P(Y = j)}{P(X = x)} = P(Y = j | X = x)\] Divide by both sides: \[f_k(x)P(Y = k) > f_j(x)P(Y = j) \] Plug in the estimated probabilities:

\[\frac{e^{-\frac{(x - \mu_k)^2}{2\sigma}}}{(2\pi)^{\frac{1}{p}}\sigma^{\frac{1}{2}}} * \frac{n_k}{n} > \frac{e^{-\frac{(x - \mu_j)^2}{2\sigma}}}{(2\pi)^{\frac{1}{p}}\sigma^{\frac{1}{2}}} * \frac{n_j}{n} \]

Dividing by like terms: \[n_ke^{-\frac{(x - \mu_k)^2}{2\sigma}} > n_je^{-\frac{(x - \mu_j)^2}{2\sigma}} \] Take the log of both sides:

\[log(n_k) -\frac{(x - u_k)^2}{2\sigma} > log(n_j) -\frac{(x - u_j)^2}{2\sigma} \] Expanding the square and subtracting by like terms we get:

\[log(n_k) +\frac{x*u_k}{2\sigma} - \frac{u_k^2}{2\sigma} > log(n_j) +\frac{x*u_j}{2\sigma} - \frac{u_j^2}{2\sigma} \] As you can see, this boundary is linear in x.

Now let's do the same thing for QDA:

\[\frac{e^{-\frac{(x - \mu_k)^2}{2\sigma_k}}}{(2\pi)^{\frac{1}{p}}\sigma_k^{\frac{1}{2}}} * \frac{n_k}{n} > \frac{e^{-\frac{(x - \mu_j)^2}{2\sigma_j}}}{(2\pi)^{\frac{1}{p}}\sigma_j^{\frac{1}{2}}} * \frac{n_j}{n} \] Divide by like terms: \[\frac{n_ke^{-\frac{(x - \mu_k)^2}{2\sigma_k}}}{\sigma_k^{\frac{1}{2}}} > \frac{n_je^{-\frac{(x - \mu_k)^2}{2\sigma_k}}}{\sigma_k^{\frac{1}{2}}} \] Take the log of both sides:

\[log(n_k)- \frac{log(\sigma_k)}{2} -\frac{(x - \mu_k)^2}{2\sigma_k} > log(n_j)- \frac{log(\sigma_j)}{2} -\frac{(x - \mu_j)^2}{2\sigma_j} \]

In this case when we exand the square we cannot get ride of the x^2 term, so this function has quadratic decision boundaries.

In order to run QDA in r, we use the qda() function from the MASS library like we did with lda().

```
qda <- qda(Species ~ ., data=train)
#predict the test set using the lda model
predict_qda <- predict(qda, test[,-5])
predict_qda
```

```
## $class
##  [1] setosa     setosa     setosa     setosa     setosa     setosa
##  [7] setosa     setosa     setosa     setosa     setosa     setosa
## [13] setosa     setosa     versicolor versicolor versicolor versicolor
## [19] versicolor versicolor versicolor versicolor versicolor versicolor
## [25] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
##
## $posterior
##            setosa   versicolor    virginica
## 5    1.000000e+00 2.567703e-28 2.323153e-43
## 6    1.000000e+00 1.097208e-27 3.151768e-42
## 8    1.000000e+00 1.304754e-24 8.389613e-40
## 10   1.000000e+00 1.687638e-21 7.910437e-37
## 11   1.000000e+00 5.048197e-30 2.507771e-46
## 14   1.000000e+00 1.332449e-20 9.116097e-36
## 18   1.000000e+00 1.503090e-25 1.271123e-41
## 23   1.000000e+00 3.444039e-28 3.192406e-43
## 29   1.000000e+00 1.021526e-25 8.720102e-43
## 30   1.000000e+00 1.723590e-20 5.163557e-34
## 36   1.000000e+00 8.196347e-24 1.311597e-41
## 41   1.000000e+00 5.845910e-26 5.855360e-42
## 44   1.000000e+00 5.611582e-18 4.584726e-31
## 49   1.000000e+00 7.198788e-30 1.497461e-45
## 54   4.361264e-73 9.980690e-01 1.930953e-03
## 56   4.903621e-91 9.923106e-01 7.689406e-03
## 62   2.691394e-84 9.989999e-01 1.000132e-03
## 65   9.514711e-54 9.999910e-01 8.976219e-06
## 68   9.995108e-67 9.998847e-01 1.153097e-04
## 73  4.309053e-125 6.754899e-01 3.245101e-01
## 74  8.447956e-100 9.389433e-01 6.105671e-02
## 83   9.514292e-65 9.999950e-01 5.020808e-06
## 87  7.384395e-112 9.996800e-01 3.200140e-04
## 94   1.445533e-38 9.999988e-01 1.179210e-06
## 107 1.040765e-106 1.686468e-02 9.831353e-01
## 122 2.550249e-142 2.023995e-05 9.999798e-01
## 128 7.880530e-131 1.843287e-01 8.156713e-01
## 130 9.085354e-187 4.091297e-03 9.959087e-01
## 139 3.150307e-125 1.627068e-01 8.372932e-01
## 144 4.078791e-221 5.960387e-07 9.999994e-01
```

We can compare the error rate of QDA with LDA:

```
#get lda predictions
preds_qda <- predict_qda$class
#get error rate
err_rate_qda <- (1/nrow(test))*sum(preds_qda != real)
err_rate_qda
```

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

Once again we get an error rate of zero, which is very lucky.

This distinction in decision boundaries is the main difference between LDA and QDA. Machine Learning textbooks will talk about flexible vs inflexible methods, and in this case, QDA is a more flexible method, while LDA is more inflexible. Logistic Regression is considered an inflexible method as well. When to use either LDA or QDA will be discussed at the end of the post. QDA is sometimes thought of as the bridge in flexibility between inflexible methods like Logistic Regression and LDA and the very flexible method I will talk about next – K Nearest Neighbors (KNN).

KNN

So far, the methods I have discussed are known as parametric methods for classification. These methods predict classification using estimated parameters: Logistic Regression estimates coefficients; LDA and QDA estimate the group mean, standard deviation, and class frequency. K - Nearest Neighbors (KNN) is a non-parametric method. No assumptions are made on the underlying distribution of the data and no parameters are estimated in order to fit a classification model.

KNN classifies a data point by comparing similarities between the given data point and the training data set. A KNN model looks at the k most similar individuals in the training data, counts the frequency of each group in the k most similar individuals, and classifies the data point as in the group with the largest representation in the k nearest neighbors.

As an example say I have 100 data points (observations/individuals) in my training data set, so I have 100 rows in my data frame. Each of the observations (rows) has a color column (attribute) that classifies them as one of “green”, “blue” or “red.” Now say I am using a KNN model with K = 5 (so 5 closest neighbors), and I want to classify an observation with an unknown color attribute. Say of the 5 most similar data points, 3 are green and 2 are blue. In this scenario I would classify my data point as “green” using the 5 nearest neighbors. Now say, using the same training data set and same data point, I want use a KNN model with K = 10, so the 10 nearest neighbors. Say of the 10 most similar data points, 4 are green, 5 are blue, and 1 is red. Now I would classify this data point as “blue.”

Using KNN as a classification tool is easy in r using the function knn() from the package “class.” The knn function takes in a training (train) and test (test) data set that do not include classification classes. The knn function also had a “cl” argument that is a factor of true classification for the training set. Finally the knn function takes a “k” argument for the number of nearest neighbors to look at in classification.

```
#install the package
#NOTE: Please install this package by running the following line on your console!!!!!
install.packages("class")
```

```
#load package class
library(class)
```

Below I provide an example of using KNN on the iris data set. The knn function returns a factor vector of predicted classes train the test data set.

```
#set seed for reproducibility
set.seed(500)
#get randome sample of 120 training index numbers
train_idx <- sample(nrow(iris), 120)
#create training set
train_set <- iris[train_idx, ]
#create set set
test_set <- iris[-train_idx, ]
#train is training set without class
train <- train_set[, -5]
#test is test set without class
test <- test_set[, -5]
#cl is true classification of the training set
cl <- train_set$Species
#knn with k = 1
knn_pred_one <- knn(train = train, test = test, cl = cl, k = 1)
knn_pred_one
```

```
##  [1] setosa     setosa     setosa     setosa     setosa     setosa
##  [7] setosa     setosa     setosa     setosa     versicolor versicolor
## [13] versicolor versicolor versicolor versicolor versicolor versicolor
## [19] versicolor virginica  versicolor versicolor versicolor virginica
## [25] versicolor virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

```
#knn with k = 5
knn_pred_five <- knn(train = train, test = test, cl = cl, k = 5)
knn_pred_five
```

```
##  [1] setosa     setosa     setosa     setosa     setosa     setosa
##  [7] setosa     setosa     setosa     setosa     versicolor versicolor
## [13] versicolor versicolor versicolor versicolor versicolor versicolor
## [19] versicolor virginica  versicolor versicolor versicolor virginica
## [25] virginica  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

We can see that the knn model with k equal to 1 and the knn model with k equal to 5 are very similar. In this case they only classified one observation differently.

```
#check how many different classifications for observations knn 1 with knn 5
sum(knn_pred_five != knn_pred_one)
```

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

We can check which one was more accurate as well.

```
#real classification
classes <- test_set$Species
#check error rate of knn 1
(1/30)*sum(classes != knn_pred_one)
```

```
## [1] 0.06666667
```

```
#check error rate of knn 5
(1/30)*sum(classes != knn_pred_five)
```

```
## [1] 0.03333333
```

As you can see, in this case knn 5 was more accurate.

KNN is probably the most simple classification model, but you may wonder how we measure similarity? In general we use Euclidean distances. Euclidean distances are also sometimes referred to as the l2-norm, and it is the square root of the dot product of two vectors. You can think of each row of a data set as a vector in p dimensions with p equal to the number of predictors. In this case p is also the number of columns in our data frame, not including the column of classifications. Two observations are considered similar if the Euclidean distance between the two vectors is small. Below is a definition of Euclidean distance, in this case p-dimensional means each vector has p items.

\[Given \ vector \ v \ and \ w \ are \ p-dimensional \ then \ the \ Euclidean \ distance \ of \ v \ to \ w \ is:\] \[d(v, w) = d(w, v) = \sqrt{\sum_{i = 1}^{p}{(v_i - w_i)^2}} \] Before, I mentioned that statistical learning models are often considered flexible or inflexible. KNN is the most flexible method I have discussed so far. Later I will go into what that means for performance compared to the other methods mentioned. You may have wondered how we decide the correct value of k for our KNN model. The value k is known as a hyper-parameter, and is generally determined via a process called cross-validation (CV), one of the most powerful ways to test the performance of a machine learning method. In terms of flexibility, a KNN model is most flexible with k equal to 1, and decreases in flexibility as the value of k increases. The implications of this will be discussed later.

Finally, I should mention one huge draw back of knn known as the curse of dimensionality. Before I mentioned that each predictor acted as a dimension for an observation, where each observation (row of our data frame) was a vector. You can imagine each observation is a point in a p-dimensional space. The issue for knn is that has the value of p increases, the knn predictors may have trouble finding “similar” data points. To see this let me present you with as example.

Imagine you have 100 numbers between 0 and 10, where each number is an observation from a uniform distribution between 0 and 1, and each number has an associated response class. Say you wish to predict the class of a data point between 0 and 1 and you can look at the data within 10 percent of your axis. On average, what proportion of the data will be in your search range? In this case the answer is 0.1, or 10 percent of the data in our search range.

Now imagine 100 observations have two features, X1 and X2, where both are an observation from a uniform distribution between 0 and 1. Now, when looking at similar data to classify your data point you can look at data within 10 percent of the X1 axis point and 10 percent of the X2 axis point. For example say the data point we are trying to classify has X1 = 0.3 and X2 = 0.7, the range of similar items we can look at is in X1 = [0.25, 0.35] and X2 = [0.65, 0.75]. Let’s see on average what proportion of the data will be in our search range. To do this just calculate the area of the search range = 0.01. We can see this time, on average, 0.01, or 1 percent of the data will be in our search range. With 100 data points, on average we only have 1 “similar” data point in 2 dimensions.

Now repeat the experiment with p = 100 features, so we have 100 dimensions. You can see now have an average proportion of (.1)^(-100), or (10)^(-98) percent of the data, in our search range. This means with 100 data points on average we never have a similar observation to our data point in 100 dimensions. This shows how difficult it can be to find “similar” observations to a data point when the data has many features.

When to use certain classification methods

Before I go over when to use each classification method I should go over some general terms in machine learning. I have mentioned that statistical learning methods are usually considered “flexible” or “inflexible,” but I haven’t explained what that means. The meaning behind these terms lies in something known as the “bias-variation trade off”. I will skip the math behind the bias-variance trade off, but the idea is the performance of a statical learning model (often measured via the mean-squared error) is a function of the variance and squared bias of that model. In general, a statistical learning method will have bias if underlying assumptions about the data are used to formulate the model, but these assumptions are not accurate. A model will have high variance if it “overfits” the model to the training data, or if multicollinearity issues are present. Don’t worry about what exactly some of those terms mean, but what’s important is that bias and variance are generally at odds, and a model that has very low bias will often suffer from large variance, and vice versa. A very flexible model will be one that has no (or next to no) bias, but may suffer from high variance. On the other hand an inflexible model reduces variance, but often at the cost of bias. You may see online that generally flexible models will perform better than inflexible models, but in reality it is based on what is best suited for the current situation.

Now I will describe the attributes of each method I have gone over and when to use each methods.

Logistic Regression

- Classification is based on P(Y = Class k | X= x)
- P(Y = Class k | X = x) is estimated using coefficients found via the Newton-Raphson algorithm
- Logistic Regression makes no assumptions on the underlying distribution of the data
- Logistic Regression is considered an inflexible method
- Logistic Regression has linear decision boundaries
- Logistic Regression performs well when there is a linear relationship between the response and data
- Logistic Regression will outperform LDA if the underlying data is not approximately normal
- Logistic Regression will often not perform well when there are more than 2 groups of classification

LDA

- Classification is based on P(Y = Class k | X= x)
- P(Y = Class k | X = x) is found using Bayes Formula
- The data is assummed to be approximately normal
- LDA is considered an inflexible method
- LDA has linear decision boundaries
- LDA performs well if there is a linear relationship between the response and data
- LDA performs well if the data truly is approximately normal
- LDA will not be negatively affected if the number of predictors is high
- LDA is preferred over logistic regression when the number of classes is greater than 2.

- LDA well not perform well if the distribution of the data is not approximately normal or if there is not a linear relationship between the predictors and response.

QDA

- QDA classification is the same as LDA, except that for each class a different estimated variance-covariance matrix (or standard deviation in the one dimensional case) is used.
- QDA has quadratic decision boundaries
- QDA is the middle ground in flexibility between inflexible methods like LDA and flexible methods like KNN
- QDA will not be negatively affected if the number of predictors is high
- QDA will not be negatively affected if the number of observations is low
- QDA will perform the best if the data is approximately normal, but collinearity issues affect the LDA performance, or the true relationship between the data and response is not linear.

KNN

- KNN is a non-parametric method of classification that is based on Euclidean distances
- KNN is a very flexible method
- KNN performs best if the data is non-Gaussian or the relationship between the data and response is highly non-linear.
- KNN may suffer from overfitting

Conclusion

In this post I introduced four of the most popular classification methods in machine learning, and described how to use each method with functions from different packages in r. I then described the pros and cons of each method, and gave a guide as to when each method should be used. In general, when formulating a statistical learning model, one should compare the performance of many different types of methods, and pick the best model for each situation. I hope you were able to learn a bit about classification in r, and have the knowledge to learn more about machine learning in r.

References

Anderson, Edgar. Iris Data. 1935. Web. https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/iris.html

Fisher, R. A. “The Use of Multiple Measurements in Taxonomic Problems”. Annals of Eugenics. 1936.

Hastie, Trevor J., et al. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2017.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An Introduction to Statistical Learning: With Applications in R. New York: Springer, 2014. Print.

ISLR: Data for an Introduction to Statistical Learning with Applications in R. R package version 1.2. https://CRAN.R-project.org/package=ISLR

S&P Stock Market Data. The Standard & Poor’s 500 Index. RDocumentation. Web. https://www.rdocumentation.org/packages/ISLR/versions/1.2/topics/Smarket

Venables, W. N. & Ripley, B. D. Package ‘class’. Cran, 2015. Web. https://cran.r-project.org/web/packages/class/class.pdf

Venables, W. N. & Ripley, B. D. Package ‘MASS’. Cran, 2017. Web. https://cran.r-project.org/web/packages/MASS/MASS.pdf

Links to references

Anderson, Edgar

Fisher, R. A

Hastie, Trevor J., et al

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani

ISLR: Data for an Introduction to Statistical Learning with Applications in R

S&P Stock Market Data

Venables, W. N. & Ripley, B. D. Package ‘class’

Venables, W. N. & Ripley, B. D. Package ‘MASS’