Chapter 03 - Classification - 2

March 27, 2022

1 Classification (02)

In this lab we would be going through: - Naive Bayes - K-Nearest Neighbours - Poisson Regression

For this lab, we would examining the `Smarket` data set that contains a number of numeric variables plus a variable called `Direction` which has the two labels `Up` and `Down`

Our goal is to predict `Direction` using the other features

[1]:
```
library(e1071)
library(ISLR2)
attach(Smarket)
```

[2]:
```
train <- (Year < 2005)

# Test data
Smarket.test <- Smarket[!train, ]
dim(Smarket.test)

#Train data
Smarket.train = Smarket[train, ]
dim(Smarket.train)

Direction.2005 = Direction[!train]
```

1. 252 2. 9

1. 998 2. 9

1.1 Naive Bayes

We are using the `naiveBayes()` function, which is part of the e1071 naiveBayes() library.

By default, this implementation of the naive Bayes classifier models each quantitative feature using a Gaussian distribution. However, a kernel density method can also be used to estimate the distributions.

[3]:
```
nb.fit <- naiveBayes(Direction ~ Lag1 + Lag2, data = Smarket,
subset = train)
```

```
nb.fit
```

```
Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
    Down       Up
0.491984 0.508016

Conditional probabilities:
      Lag1
Y              [,1]     [,2]
  Down  0.04279022 1.227446
  Up   -0.03954635 1.231668

      Lag2
Y              [,1]     [,2]
  Down  0.03389409 1.239191
  Up   -0.03132544 1.220765
```

The output contains the estimated mean and standard deviation for each variable in each class.

[4]:
```
mean(Lag1[train][Direction[train] == 'Down'])
sd(Lag1[train][Direction[train] == 'Down'])
```

0.0427902240325866

1.22744562820108

[5]:
```
nb.class = predict(nb.fit, Smarket.test)
table(nb.class, Direction.2005)

mean(nb.class == Direction.2005)
```

```
        Direction.2005
nb.class Down  Up
    Down   28  20
    Up     83 121
```

0.591269841269841

`Naive Bayes` performs very well on this data, with accurate predictions over `59%` of the time. This is slightly worse than `QDA`, but much better than `LDA`.

The `predict()` function can also generate estimates of the probability that each observation belongs to a particular class

[6]:
```
nb.preds = predict(nb.fit, Smarket.test, type = "raw")
nb.preds[1:5, ]
```

A matrix: 5 × 2 of type dbl

Down	Up
0.4873164	0.5126836
0.4762492	0.5237508
0.4653377	0.5346623
0.4748652	0.5251348
0.4901890	0.5098110

1.2 K - Nearest Neighbors

We would be using the `knn()` function which is a part of the `class` library. Rather than a two-step approach in which we first fit the model and then we use the model to make predictions, `knn()` forms predictions using a single command.

The function requires four inputs: 1. A matrix containing the predictors associated with the training data, labeled `train.X` below. 2. A matrix containing the predictors associated with the data for which we wish to make predictions, labeled `test.X` below. 3. A vector containing the class labels for the training observations, labeled `train.Direction` below. 4. A value for `K`, the number of nearest neighbors to be used by the classifier.

[7]:
```
library(class)
```

[8]:
```
train.X = cbind(Lag1, Lag2)[train, ] #cbind() is short for column bind, binds␣
↪variables together
test.X = cbind(Lag1, Lag2)[!train, ]
train.Direction = Direction[train]
```

We set a random `seed` before we apply `knn()` because if several observations are tied as nearest neighbors, then `R` will randomly break the tie.

[9]:
```
set.seed(1)
knn.pred = knn(train.X, test.X, train.Direction, k=1)

table(knn.pred, Direction.2005)
mean(knn.pred==Direction.2005) #performance
```

```
        Direction.2005
knn.pred Down Up
    Down   43 58
    Up     68 83
```

0.5

The results using `K = 1` are not very good, since only `50 %` of the observa- tions are correctly predicted. Of course, it may be that `K = 1` results in an overly flexible fit to the data.

[12]:
```
#return a k-nn model with three neighbors
knn.pred = function(){
    # your code here
    return(knn(train.X, test.X, train.Direction, k=3))
}
knn.pred = knn.pred()
```

[13]:
```
table(knn.pred, Direction.2005)

#Test the performance of new model
stopifnot(round(mean(knn.pred == Direction.2005),2) == 0.54)
```

```
        Direction.2005
knn.pred Down Up
    Down   48 54
    Up     63 87
```

[14]:
```
knn.pred = knn(train.X, test.X, train.Direction, k=4)
mean(knn.pred == Direction.2005)
```

0.496031746031746

We can see that the results have improved slightly when we increase the value of `K` from `1` to `3`. But increasing `K` further turns out to provide no further improvements.

It appears that for this data, QDA provides the best results of the methods that we have examined so far.

1.3 Poisson Regression

We would be using the `glm()` function with the argument `family = poisson` to define a poisson regression model.

We are gonna fit a Poisson regression model to the `Bikeshare` data set found in `ISLR2` library, which measures the number of bike rentals(`bikers`) per hour in Washington DC.

[15]:
```
attach(Bikeshare) #attaching the data set to R's context
```

[16]:
```
dim(Bikeshare)
names(Bikeshare)
```

1. 8645 2. 15

1. 'season' 2. 'mnth' 3. 'day' 4. 'hr' 5. 'holiday' 6. 'weekday' 7. 'workingday' 8. 'weathersit' 9. 'temp' 10. 'atemp' 11. 'hum' 12. 'windspeed' 13. 'casual' 14. 'registered' 15. 'bikers'

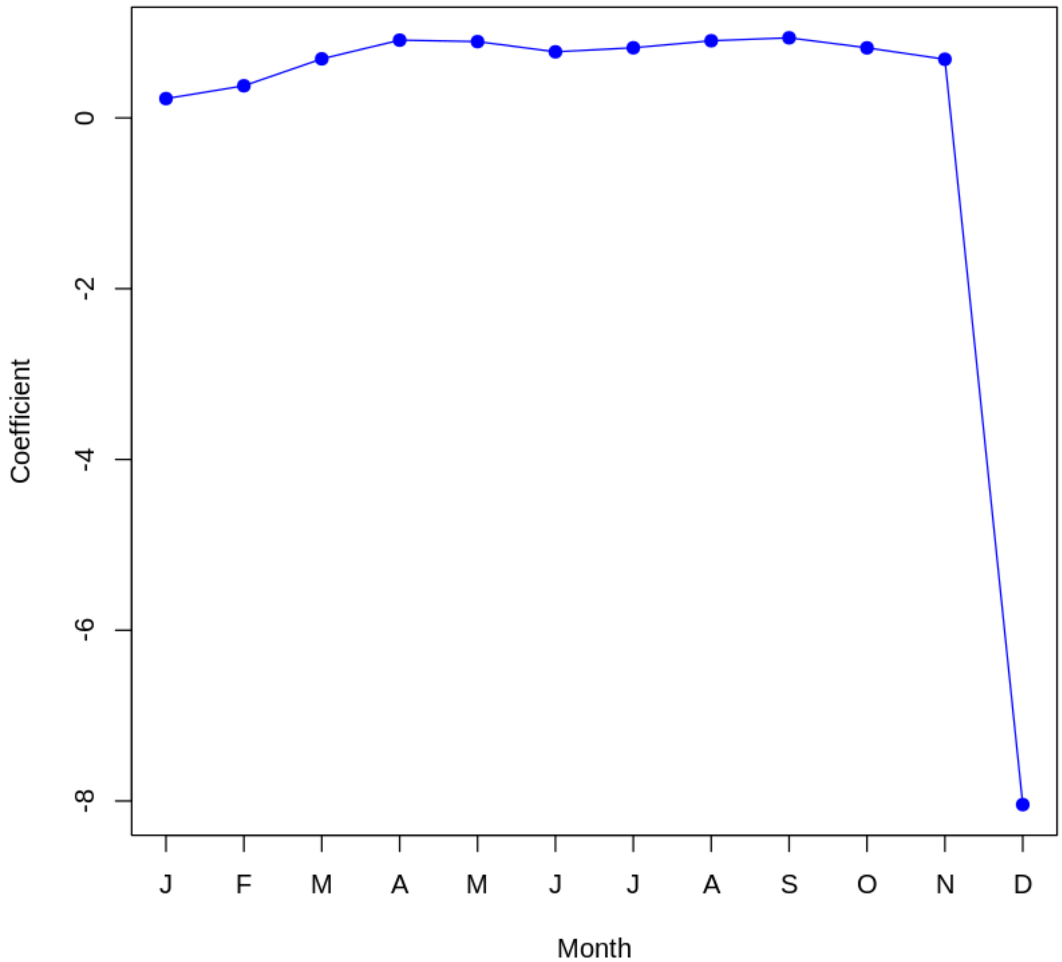
[17]:
```r
mod.pois = glm(bikers ~ mnth + hr + workingday + temp + weathersit,
               data = Bikeshare, family = poisson)
summary(mod.pois)
```

```
Call:
glm(formula = bikers ~ mnth + hr + workingday + temp + weathersit,
    family = poisson, data = Bikeshare)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-20.7574   -3.3441   -0.6549    2.6999   21.9628

Coefficients:
                           Estimate Std. Error  z value Pr(>|z|)
(Intercept)                2.693688   0.009720  277.124  < 2e-16 ***
mnthFeb                    0.226046   0.006951   32.521  < 2e-16 ***
mnthMarch                  0.376437   0.006691   56.263  < 2e-16 ***
mnthApril                  0.691693   0.006987   98.996  < 2e-16 ***
mnthMay                    0.910641   0.007436  122.469  < 2e-16 ***
mnthJune                   0.893405   0.008242  108.402  < 2e-16 ***
mnthJuly                   0.773787   0.008806   87.874  < 2e-16 ***
mnthAug                    0.821341   0.008332   98.573  < 2e-16 ***
mnthSept                   0.903663   0.007621  118.578  < 2e-16 ***
mnthOct                    0.937743   0.006744  139.054  < 2e-16 ***
mnthNov                    0.820433   0.006494  126.334  < 2e-16 ***
mnthDec                    0.686850   0.006317  108.724  < 2e-16 ***
hr1                       -0.471593   0.012999  -36.278  < 2e-16 ***
hr2                       -0.808761   0.014646  -55.220  < 2e-16 ***
hr3                       -1.443918   0.018843  -76.631  < 2e-16 ***
hr4                       -2.076098   0.024796  -83.728  < 2e-16 ***
hr5                       -1.060271   0.016075  -65.957  < 2e-16 ***
hr6                        0.324498   0.010610   30.585  < 2e-16 ***
hr7                        1.329567   0.009056  146.822  < 2e-16 ***
hr8                        1.831313   0.008653  211.630  < 2e-16 ***
hr9                        1.336155   0.009016  148.191  < 2e-16 ***
hr10                       1.091238   0.009261  117.831  < 2e-16 ***
hr11                       1.248507   0.009093  137.304  < 2e-16 ***
hr12                       1.434028   0.008936  160.486  < 2e-16 ***
hr13                       1.427951   0.008951  159.529  < 2e-16 ***
hr14                       1.379296   0.008999  153.266  < 2e-16 ***
hr15                       1.408149   0.008977  156.862  < 2e-16 ***
hr16                       1.628688   0.008805  184.979  < 2e-16 ***
hr17                       2.049021   0.008565  239.221  < 2e-16 ***
hr18                       1.966668   0.008586  229.065  < 2e-16 ***
hr19                       1.668409   0.008743  190.830  < 2e-16 ***
hr20                       1.370588   0.008973  152.737  < 2e-16 ***
hr21                       1.118568   0.009215  121.383  < 2e-16 ***
```

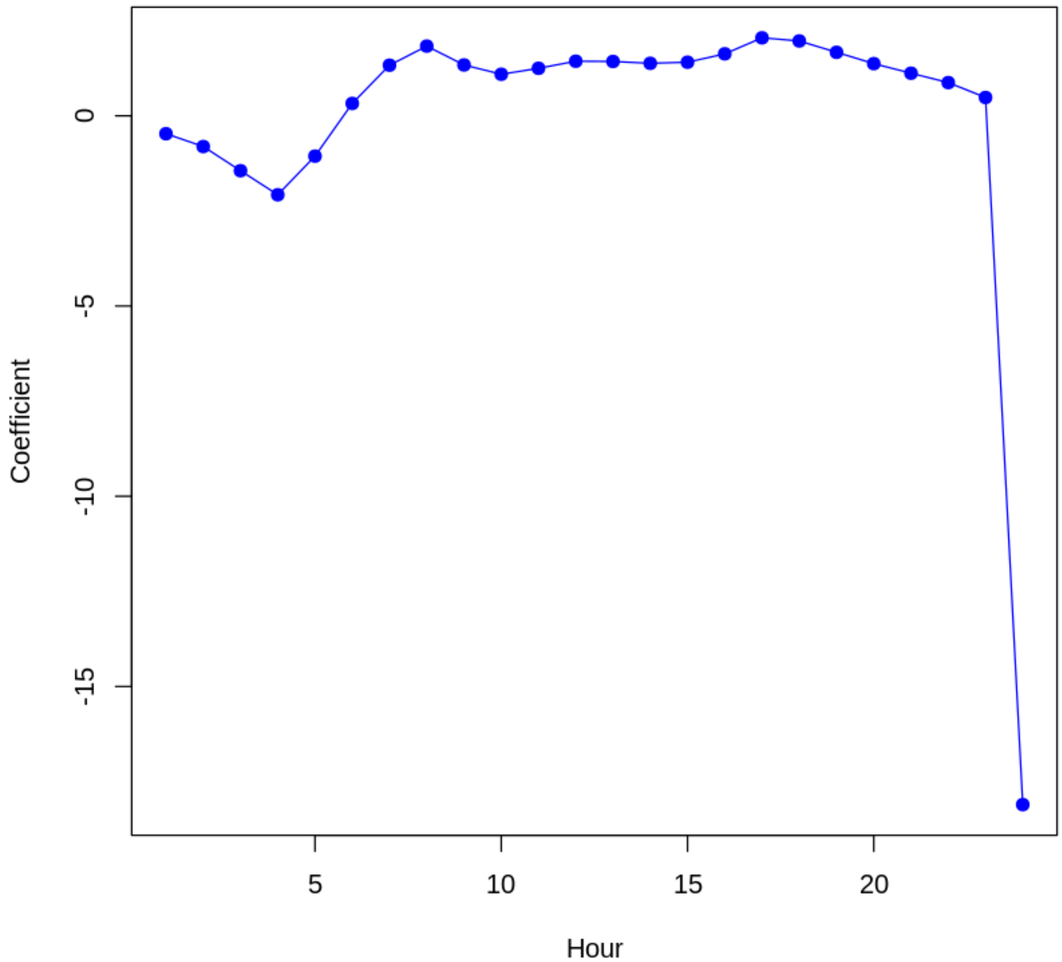
```
hr22                          0.871879   0.009536   91.429  < 2e-16 ***
hr23                          0.481387   0.010207   47.164  < 2e-16 ***
workingday                    0.014665   0.001955    7.502 6.27e-14 ***
temp                          0.785292   0.011475   68.434  < 2e-16 ***
weathersitcloudy/misty       -0.075231   0.002179  -34.528  < 2e-16 ***
weathersitlight rain/snow -0.575800   0.004058 -141.905  < 2e-16 ***
weathersitheavy rain/snow -0.926287   0.166782   -5.554 2.79e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 1052921  on 8644  degrees of freedom
Residual deviance:  228041  on 8605  degrees of freedom
AIC: 281159

Number of Fisher Scoring iterations: 5
```

We are gonna plot these coefficients associated with **mnth** and **hr** for better visualization

[18]:
```r
coef.mnth <- c(coef(mod.pois)[2:12], -sum(coef(mod.pois)[2:12]))



plot(coef.mnth, xlab = "Month", ylab = "Coefficient",
     xaxt = "n", col = "blue", pch = 19, type = "o")
axis(side = 1, at = 1:12,
     labels = c("J", "F", "M", "A", "M", "J", "J", "A", "S", "O", "N", "D"))

coef.hours <- c(coef(mod.pois)[13:35], -sum(coef(mod.pois)[13:35]))
plot(coef.hours, xlab = "Hour", ylab = "Coefficient", col = "blue", pch = 19,␣
 ↪type = "o")
```



Coefficient
0
-2
-4
-6
-8
J
F
M
A
M
J
J
A
S
O
N
D
Month



We can once again use the `predict()` function to obtain the fitted values (predictions) from this Poisson regression model.

[19]:
```
mod.pred = predict(mod.pois, type = "response")
summary(mod.pred)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  1.201  44.513 124.299 143.794 219.268 585.958 
```

[ ]:
```
```