Problem Set 8, Fall 2021

Ben Karabinus

```
# Load any packages, if any, that you use as part of your answers here
# For example:

library(MASS)
library(glmnet)

## Loading required package: Matrix

## Loaded glmnet 4.1-2

library(mlbench)
library(survival)
library(survminer)

## Loading required package: ggplot2

## Loading required package: ggpubr
```

CCONTEXT - HOUSE VALUES IN BOSTON, CIRCA 1970

This dataset was obtained through the mlbench package, which contains a subset of data sets available through the UCI Machine Learning Repository. From the help file:

Housing data for 506 census tracts of Boston from the 1970 census. The dataframe BostonHousing contains the original data by Harrison and Rubinfeld (1979).

The original data are 506 observations on 14 variables, medv being the target variable:

Continuous variables:

crim per capita crime rate by town zn proportion of residential land zoned for lots over 25,000 sq.ft
indus proportion of non-retail business acres per town nox nitric oxides concentration (parts per 10 million) rm average number of rooms per dwelling age proportion of owner-occupied units built prior to 1940 dis weighted distances to five Boston employment centres rad index of accessibility to radial highways tax full-value property-tax rate per USD 10,000 ptratio pupil-teacher ratio by town b 1000(B - 0.63)^2 where B is the proportion of blacks by town lstat percentage of lower status of the population medv median value of owner-occupied homes in USD 1000's

Categorical variables:

chas Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)

Question 1 - 10 points

First, load the data into memory. The variable types are already stored in this data set.

```
data(BostonHousing) # loads the BostonHousing dataset into memory from the ml
bench package

str(BostonHousing)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.5
24 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : num  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ b      : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

Before you begin your analysis, you will split the data into a 70% training set and a 30% test set. First, save the number of rows in the data set for use in the splitting code.

```
# get the number of rows
n <- nrow(BostonHousing)
```

When splitting data into training/test data sets, it's good practice to set a random seed to create a split that's reproducible. For this question, use the following seed.

```
# set seed for reproducability
set.seed(123456)
```

In Problem Set 6, you were shown some code from the async to create a train-validate-test split

tvt2 <- sample(rep(0:2,c(round(n.*2),round(n*.2),n-2*round(n*.2))),n)

In this problem, however, you are splitting your data into just training and test sets (i.e., just two groups). You can make some changes to the rep() function contained in this line code to create a split for just train-test. To help you make these adaptations, the following code chunk contains the isolated version of what's contained in the tvt2 rep() function. Run it to see what it produces and then make alterations that will instead produce a set of 0's (test set, 30%) and 1's (training set, 70%) for splitting purposes.

```
tvt2.rep <- rep(0:2,c(round(n*.2),round(n*.2),n-2*round(n*.2))) # The .2 in t
his function produces a 80% train/20% validation/20% test split in the data

tvt2.rep # Shows the result in the console window

##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [223] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [260] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [297] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [334] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [371] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [408] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [445] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
## [482] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

table(tvt2.rep) # Shows a count of the 0's (test), 1's (valid), and 2's (trai
n)

## tvt2.rep
##   0   1   2
## 101 101 304

# Here is some room for you to change things and test how they work
# tvt3.rep for testing purposes
tvt3.rep <- sample(rep(0:1,c(round(n*.3),n-round(n*.3))),n)
# table tvt3.rep
table(tvt3.rep)

## tvt3.rep
##   0   1
## 152 354
```

Once you've found something that works, insert it into the blank space in tv.split to obtain a 70% training/30% test split. Display a table of the split to verify that approximately 70% of tv.split is equal to 1 and approximately 30% is equal to zero

```
# set seed for reproducability
set.seed(123456)
# create random sample of int 0-1 from n, proportions .70, .30
tv.split <- sample(rep(0:1,c(round(n*.3),n-round(n*.3))),n)
# print table of counts (0, 1) for verification
table(tv.split)

## tv.split
##   0   1
## 152 354

# training set (352)
dat.train <- BostonHousing[tv.split==1,]
# test set (152)
dat.test <- BostonHousing[tv.split==0,]
```

Question 2 - 10 points

After completing Question 1, conduct a cross-validated ridge regression using the training data set. Use medv as the outcome and all of the other variables in the data set as the predictors.

```
# create the matrix of explanatory variables
X <- model.matrix(medv ~., data=dat.train)
# drop the intercept column
X <- X[,-1]
# create the vector of outcomes
Y<-dat.train$medv
# set seed for reproducability
set.seed(123456)
# create cross validated ridge regression model (alpha = 0 for ridge)
cvfit.house.ridge <- cv.glmnet(x=X, y=Y,alpha=0)
```

For this question, the only lambda of interest is lambda.min. Make sure that lambda.min and the coefficients associated with it are visible in your knitted document.

```
# print the smallest lambda for the fitted model
cvfit.house.ridge$lambda.min

## [1] 0.6807801

# print coefficients associated with smallest lambda
coef(cvfit.house.ridge, s = "lambda.min")

## 14 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept)  30.021894484
## crim         -0.100660717
## zn              0.026631639
## indus          -0.066953316
## chas1           3.210952695
## nox           -12.070879920
## rm              4.018937802
## age            -0.003031209
## dis            -1.141512339
## rad             0.158816580
## tax            -0.004886745
## ptratio        -0.948040004
## b               0.009192364
## lstat          -0.469755029
```

Question 3 - 5 points

Using the results from Question 2, compute the mean squared prediction error for the lambda.min model when applied to the *test* data set. Be sure to show how you computed it and to display the result; once you've done that, answer the question below.

```
# create matrix of predictors
Xtest <- model.matrix(medv ~., data=dat.test)
# drop intercept
Xtest <- Xtest[,-1]
# vector of outcomes for prediction
Ytest<-dat.test$medv
# create prediction
cv.house.pred<-predict(cvfit.house.ridge,Xtest,c(cvfit.house.ridge$lambda.min
))
# calculate and print MSPE
MSPE <- mean((cv.house.pred-Ytest)^2)
MSPE

## [1] 22.83288
```

A) What is the mean squared prediction error you computed (your answer here):

22.83288

CONTEXT - NYC BIKERS

The NYC Open Data Portal contains information about the number of cyclists who cross different bridges in the eastern part of New York City. The data for this question is an edited subset of the data available. To see the full data, see https://data.cityofnewyork.us/Transportation/Bicycle-Counts-for-East-River-Bridges/gua4-p9wg.

Variables of interest for this question (all are continuous):

M_bridge_count: The daily count of cyclists who ride across the Manhattan Bridge temp_hi: The highest temperature recorded that day (in Fahrenheit) precipitation: The amount of precipitation recorded that day (in inches)

Question 4 - 15 points

The outcome of interest in this analysis is M_bridge_count. If you look at the values in this variable, you will see that the values contain commas to mark the thousandths place. First, remove the commas using any method, then demonstrate that the commas have been removed by displaying the first few values of the cleaned variable the head() function

```
# load the data
bike<-read.csv("NYCBikes.csv")
# remove the commas from M_bridge_count
bike$M_bridge_count = gsub("[\\,]", "", bike$M_bridge_count)
# print head to examine results
head(bike$M_bridge_count)

## [1] "1446" "3943" "4988" "1913" "5276" "1324"
```

Your two predictor variables, temp_hi and precipitation, should already be numeric. Your cleaned outcome variable, however, may need to be re-typed as a numeric variable.

```
# Code to re-type your cleaned M_bridge_count variable
bike$M_bridge_count <- as.numeric(bike$M_bridge_count)
# Display with the str function to verify that variables are correctly typed
str(bike)

## 'data.frame':    83 obs. of  9 variables:
##  $ date          : chr  "1-Apr" "2-Apr" "3-Apr" "4-Apr" ...
##  $ day           : chr  "Saturday" "Sunday" "Monday" "Tuesday" ...
##  $ temp_hi       : num  46 62.1 63 51.1 63 48.9 55.9 66 73.9 80.1 ...
##  $ temp_low      : num  37 41 50 46 46 41 39.9 45 55 62.1 ...
##  $ precipitation : num  0 0 0.03 1.18 0 0.73 0 0 0 0 ...
##  $ B_bridge_count: chr  "606" "2,021" "2,470" "723" ...
##  $ M_bridge_count: num  1446 3943 4988 1913 5276 ...
##  $ W_bridge_count: chr  "1,915" "4,207" "5,178" "2,279" ...
##  $ Q_bridge_count: chr  "1,430" "2,862" "3,689" "1,666" ...

# drop unnecessary variables
bike.dat <- dplyr::select(bike, temp_hi, precipitation, M_bridge_count)
```

```r
# check data structure
str(bike.dat)
```

```
## 'data.frame':    83 obs. of  3 variables:
##  $ temp_hi       : num  46 62.1 63 51.1 63 48.9 55.9 66 73.9 80.1 ...
##  $ precipitation : num  0 0 0.03 1.18 0 0.73 0 0 0 0 ...
##  $ M_bridge_count: num  1446 3943 4988 1913 5276 ...
```

Now you will fit three models using this data: a Poisson model, a quasipossion model, and a negative binomial model. The outcome of these analyses should be M_bridge_count, and the predictors should be temp_hi and precipitation.

Poisson model

```r
# create the Poisson Regrssion model
model.poisson <- glm(M_bridge_count~temp_hi+precipitation, data=bike.dat,
                     family ='poisson')
# print model summary
summary(model.poisson)
```

```
##
## Call:
## glm(formula = M_bridge_count ~ temp_hi + precipitation, family = "poisson"
,
##     data = bike.dat)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -42.87  -15.15   -0.94   13.35   31.00
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    7.247542   0.010796   671.3   <2e-16 ***
## temp_hi        0.019148   0.000147   130.2   <2e-16 ***
## precipitation -0.667930   0.005921  -112.8   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 71103  on 82  degrees of freedom
## Residual deviance: 29149  on 80  degrees of freedom
## AIC: 30006
##
## Number of Fisher Scoring iterations: 4
```

Quasipoisson model

```r
# create the Quasipoisson Regression model
model.quasipoisson <- glm(M_bridge_count~temp_hi+precipitation, data=bike.dat
,
                          family ='quasipoisson')
# print the model summary
summary(model.quasipoisson)

##
## Call:
## glm(formula = M_bridge_count ~ temp_hi + precipitation, family = "quasipoi
sson",
##     data = bike.dat)
##
## Deviance Residuals:
##    Min       1Q   Median       3Q      Max
## -42.87   -15.15    -0.94    13.35    31.00
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    7.247542   0.202585  35.775  < 2e-16 ***
## temp_hi        0.019148   0.002759   6.940 9.24e-10 ***
## precipitation -0.667930   0.111113  -6.011 5.21e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasipoisson family taken to be 352.1449)
##
##     Null deviance: 71103  on 82  degrees of freedom
## Residual deviance: 29149  on 80  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 4
```

Negative binomial model

```
# create the Negative Binomial model
model.nb <- glm.nb(M_bridge_count~temp_hi+precipitation, data=bike.dat)
# print model summary
summary(model.nb)

##
## Call:
## glm.nb(formula = M_bridge_count ~ temp_hi + precipitation, data = bike.dat
,
##     init.theta = 10.46595391, link = log)
##
## Deviance Residuals:
##     Min       1Q    Median       3Q      Max
## -3.3169  -0.6711  -0.0413   0.6359   2.0752
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    6.894667   0.228107  30.226  < 2e-16 ***
## temp_hi        0.023885   0.003201   7.462 8.50e-14 ***
## precipitation -0.527340   0.075537  -6.981 2.93e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(10.466) family taken to be 1)
##
##     Null deviance: 191.327  on 82  degrees of freedom
## Residual deviance:  84.447  on 80  degrees of freedom
## AIC: 1448.3
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  10.47
##           Std. Err.:  1.61
##
##  2 x log-likelihood:  -1440.292
```

Once you've fit all three models, answer the three questions below.

A) Look at the output for the Poisson model and the quasipoisson model. Which of these - Poisson or quasipoisson - have larger standard errors for the coefficients?

Your answer here (Poisson or quasipoisson): Quasipoisson

B) Look at the quasipoisson model output. What was the dispersion parameter taken to be in your model?

Your answer here: 352.1449

C) Per the guidelines presented in the async and discussed during the live session, which of the three models - Poisson, quasipoisson, and negative binomial - is the best based on the *residual deviance*?

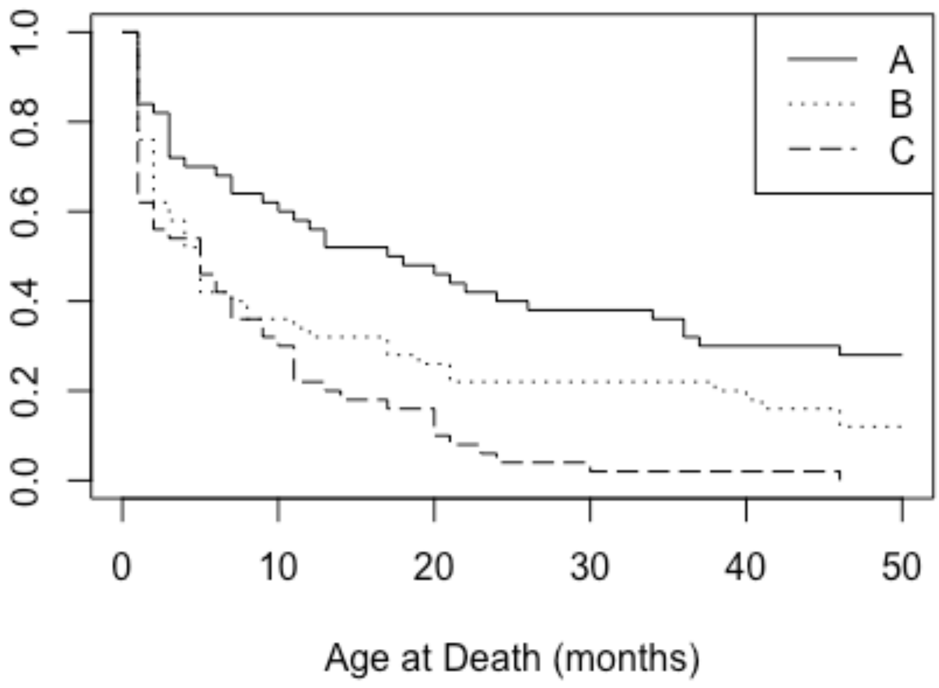
Your answer here (Poisson, quasipoisson, or negative binomial): Negative Binomial

Question 5 - 15 points

Before beginning this question, please review the material from 9.4.3 in the async material.

The following code is excerpted from the example shown in 9.4.3. The outcome of interest is time to death of sheep. Each sheep received some level of anti-parasite treatment; A and B contained actual anti-parasite ingredients and C was a placebo (i.e., no active ingredient in the treatment). Please run the three code chunks and examine their output. Once you've done that, answer the four questions below.

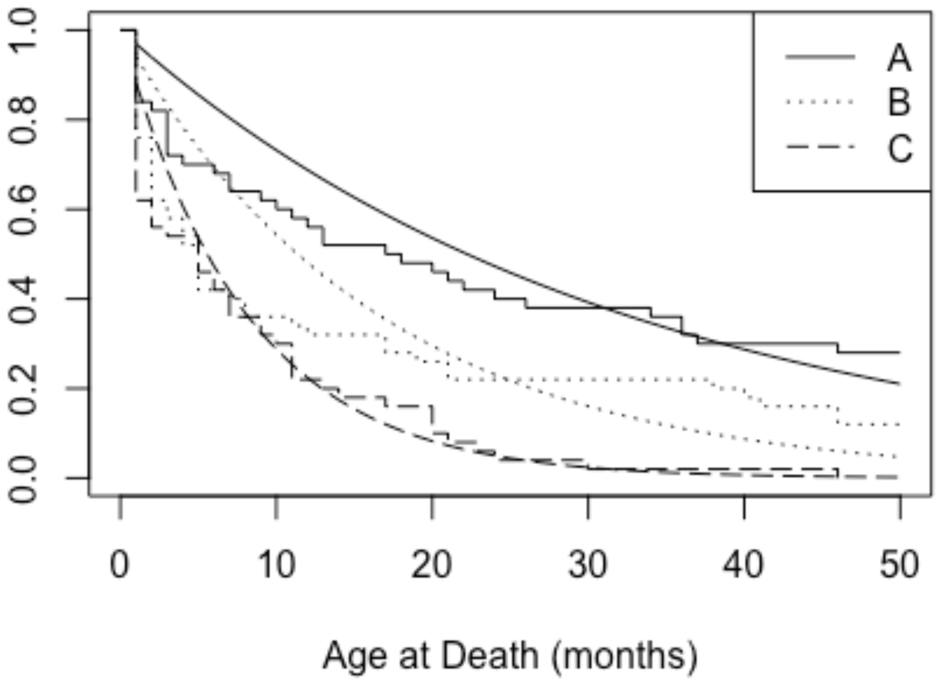
```
# Chunk 1

sheep<-read.csv("sheep.deaths.csv")

with(sheep,plot(survfit(Surv(death,status)~group),lty=c(1,3,5),xlab="Age at D
eath (months)"))
legend("topright", c("A", "B","C"), lty = c(1,3,5))
```



```
# Chunk 2

model<-survreg(Surv(death,status)~group, dist="exponential",data=sheep)
summary(model)
```

```
##
## Call:
## survreg(formula = Surv(death, status) ~ group, data = sheep,
##     dist = "exponential")
##               Value Std. Error     z       p
## (Intercept)  3.467      0.167 20.80 < 2e-16
## groupB      -0.671      0.225 -2.99  0.0028
## groupC      -1.386      0.219 -6.34 2.3e-10
##
## Scale fixed at 1
##
## Exponential distribution
## Loglik(model)= -482   Loglik(intercept only)= -502.1
##  Chisq= 40.35 on 2 degrees of freedom, p= 1.7e-09
## Number of Newton-Raphson Iterations: 5
## n= 150
```

```
# Chunk 3

plot(survfit(Surv(sheep$death,sheep$status)~sheep$group),lty=c(1,3,5),xlab="A
ge at Death (months)")
legend("topright", c("A", "B","C"), lty = c(1,3,5))

points(1:50,
       1-pexp(1:50,rate=1/exp(model$coefficients[1])),
       type="l",
      lty=1)
# The survival curve S(t) for group B.
points(1:50,
       1-pexp(1:50,rate=1/exp(sum(model$coefficients[c(1,2)]))),
       type="l",
      lty=3)
# The survival curve S(t) for group C.
points(1:50,
1-pexp(1:50,rate=1/exp(sum(model$coefficients[c(1,3)]))),
       type="l",
      lty=5)
```



Question about Chunk 1

A) What kind of plot is this? It has a specific name.

Your answer here:

The type of plot is the Kaplan-Meier plot. It shows the Kaplan-Meir curve which estimates the proportion of subjects that survive to a specific point in time in a survival analysis.

Questions about Chunk 2

B) What kind of survival model is being fitted in this code?

Your answer here:

An exponential survival model is being fit in code chunk 2.

C) What does the output of the model fitted using the survreg() function suggest about the treatment groups (A, B, and C)?

Your answer here:

The model output suggests that sheep in groups B and C have a significantly shorter lifespan than sheep in the reference category group A.

Question about Chunk 3

D) The jagged lines on this plot are the same as those from the plot shown in Chunk 1. What is being visualized by the the *smooth, curved lines* in this plot?

Your answer here:

The smoothed curved lines plotted over the Kaplan-Meier curves represent the predicted proportion of sheep still alive at a given point in time in each of the three groups A, B and C.The smoothed lines provide a visual assessment of the goodness of fit of the model.