MSIA 420 PREDICTIVE ANALYTICS
Homework 3

Shiyi Chen, Ameer Khan, Xiang Li, Luis Lin, Sanjeevni Wanchoo

***Note: in order to be consistent and be able to compare SSE across models, the response variable was scaled*

Problem 1

a. n-fold cross validation (no need to do replicates because is n-fold) were conducted for K=1 to 10 for nearest neighbor models..

k	SSE
1	391.294
2	300.53
3	257.405
4	245.136
5	233.278
6	233.66
7	224.511
8	**223.2**
9	224.008
10	224.492

k	SSE
1	17.5435
2	13.4767
3	11.5384
4	10.9977
5	10.4741
6	10.4807
7	10.0504
8	**10.0105**
9	10.0451
10	10.0677

Best model for KNN: **K = 8** has the lowest CV SSE (the table on the left shows the SSE for the log response, while the table on the right shows the SSE for the scaled log response. Note that the conclusions of the best K does not change)

CV is about estimating the prediction error. The number of folds exhibits the classic bias and variance tradeoff which describes the pros and cons of the number of folds. With n-fold more observations are used as the training set, which means that there is less bias towards overestimating the true expected error. However, there is more variance in the estimated error because there is only one observation (no replicates) in the test dataset to predict for each fold. For fewer number of folds (e.g. 10-fold), there is more bias and less variance since more observations are included in the test set at each fold and there is more overlap across folds. n-folds are usually more computationally intensive for fitting models. However, in the case of knn there is no disadvantage of using n-folds as they can have about the same computational effort as 10-fold since there is no need to run replicates compared to 10-fold.

b. The prediction error standard deviation was calculated by using two approaches: taking the square root of the CV SSE/n, or taking the standard deviation of the prediction error (difference between observed and predicted response in CV).

For unscaled:

```
> mean(sqrt(SSE[,K_opt]/n))
[1] 0.5322116
```

```
> mean(sderror[,K_opt])
[1] 0.5290012
```

For scaled:

```
> mean(sqrt(SSE[,K_opt]/n))
[1] 0.1127204
> mean(sderror[,K_opt])
[1] 0.1120404
```

c. To find the predicted cost, the values of the predictors for the new case was first standardized and then K-NN with K=3 was run to find the neighbors and average their response values to predict the cost for the new case.

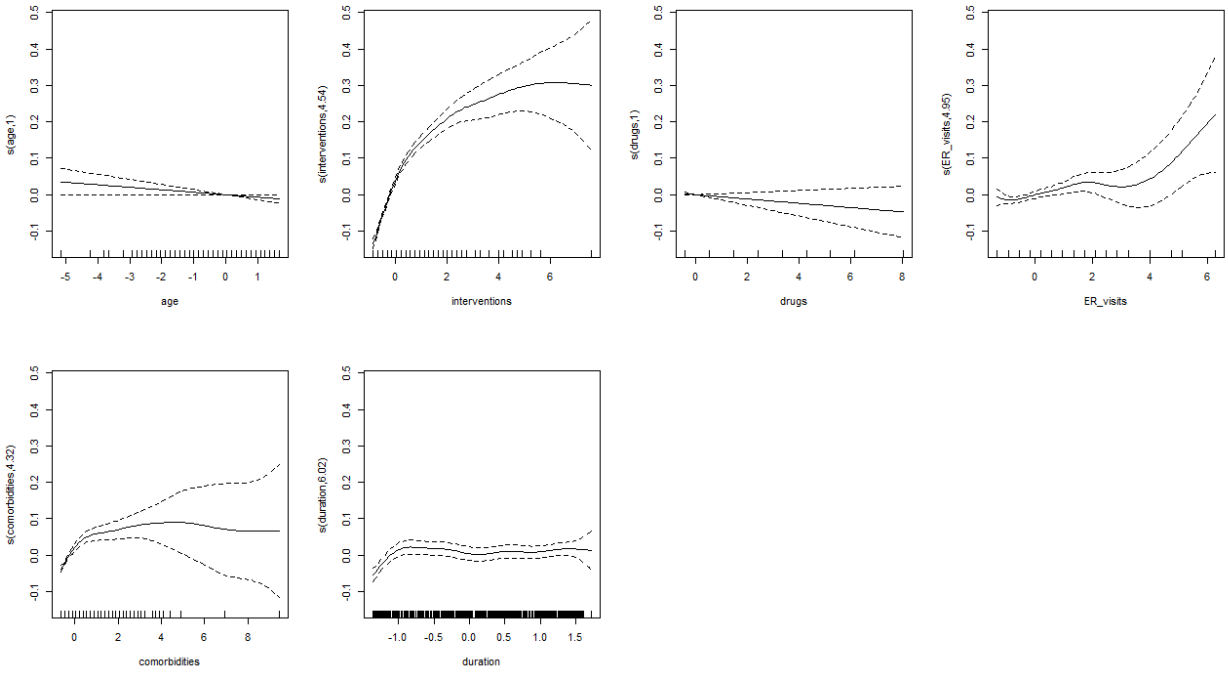
The predicted cost is: $ 4611.75 (dollar scale), 3.565623 (log scale,if we unlog this value = $ 2396.013)

Problem 2

a. A GAM model was fitted as smoothing functions of the predictors. Categorical predictors or variables that had very few distinct values(e.g. gender and complications) were fitted as linear terms in the models without smoothing functions to avoid issues of perfect collinearity and degrees of freedom.

```
> summary(out)

Family: gaussian
Link function: identity

Formula:
total_cost ~ s(age) + gender + s(interventions) + s(drugs, k = 8) +
    s(ER_visits) + complications + s(comorbidities) + s(duration)

Parametric coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.578567   0.003506 165.041  < 2e-16 ***
gender      -0.006254   0.003568  -1.753     0.08 .
complications 0.015178   0.003663   4.144  3.8e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:
                edf  Ref.df  F  p-value
```

```
s(age)              1.000  1.000    3.609        0.0578 .
s(interventions)    4.543  5.490    137.559    < 2e-16 ***
s(drugs)            1.000  1.000    1.794        0.1808
s(ER_visits)        4.948  5.969    3.154        0.0047 **
s(comorbidities  ) 4.320  5.264    15.825       1.80e-15 ***
s(duration)         6.020  7.147    5.273        5.59e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.685   Deviance explained = 69.5%
GCV score = 0.0099989  Scale est. = 0.0096839  n = 788
> out$sp  ##estimated smoothing parameters for each constituent function
      s(age          ) s(interventions)       s(drugs)    s(ER_visits)    s(comorbidities)       s(duration)
   4.512303e+06      2.256746e-03      9.778386e+05      1.795187e-03      1.390672e-03      1.429844e-03
```



The interpretation of the plots is the overall average effect (main effect, assuming no interactions) of the predictors on the response:

- As intervention increases, the cost increases the most and then plateaus
- As comorb increases, the costs slightly increases and the plateaus
- As duration increases, the costs slightly increases and the plateaus
- At high values of ervis, as ervis increase the cost increases at increasing rate
- The change of the predicted response decreases linearly with increase usage of drugs
- The change of the predicted response decreases linearly with increase age

The higher the edf , the higher complexity of smoothing function.The p-value shows age, gender and drugs are not significant. The order of significant predictors from most important to least important (based on p-values): interv, comorb, duration, complications, ervis.

b. The prediction error standard deviation was calculated by using two approaches: taking the square root of the CV SSE/n, or taking the standard deviation of the prediction error (difference between observed and predicted response in CV).

```
> mean(sqrt(SSE[,1]/n))
[1] 0.1007351
>
> # or
>
> mean(sderror[,1])
[1] 0.1007989
```

Similar answer to 1a due to the nonparametric nature of GAM. However, the advantage of about the same computational effort for n-fold and 10-fold is not as important as in nearest neighbors because a smoothing function has to be estimated and thus might take longer.

c. To find the predicted cost, the values of the predictors for the new case was first standardized and then GAM model from part a. was run to find the predicted value.

The predicted cost is: $ 3678.09 (dollar scale), 3.565623 (log scale)

Problem 3

a. For this problem, only intvn, comorb, dur, ervis were used for prediction. 10-fold cross validation was performed, and the cross-validation SSE and R-square for the kernel method was measured across several different spans (lambda in the result below) and degree = 0,1, and 2 combinations. These are shown below.

```
> results
   model lambda          sse           r2
1      0   0.01 1.263318e+01  4.861102e-01
2      0   0.03 8.141017e+00  6.658198e-01
3      0   0.05 7.967359e+00  6.738600e-01
4      0   0.07 7.984569e+00  6.738217e-01
5      0   0.10 8.020234e+00  6.735533e-01
6      0   0.30 9.180725e+00  6.299522e-01
7      0   0.50 1.055339e+01  5.748806e-01
8      0   0.70 1.199370e+01  5.161069e-01
9      1   0.01 1.285703e+02 -4.300694e+00
10     1   0.03 1.069208e+01  5.583538e-01
11     1   0.05 8.952671e+00  6.301346e-01
12     1   0.07 8.456127e+00  6.507818e-01
```

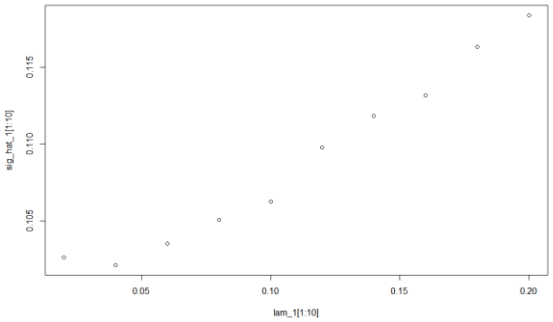
13	1	0.10	8.092862e+00	6.660089e-01
14	1	0.30	7.796821e+00	6.796743e-01
15	1	0.50	7.844271e+00	6.796216e-01
16	1	0.70	7.961082e+00	6.759999e-01
17	2	0.01	3.659678e+14	-1.510459e+13
18	2	0.03	4.476771e+02	-1.746111e+01
19	2	0.05	9.404173e+01	-2.883448e+00
20	2	0.07	1.679336e+01	3.063124e-01
21	2	0.10	1.061648e+01	5.614501e-01
22	2	0.30	8.307187e+00	6.570247e-01
23	2	0.50	8.014219e+00	6.694964e-01
24	2	0.70	7.941790e+00	6.727376e-01

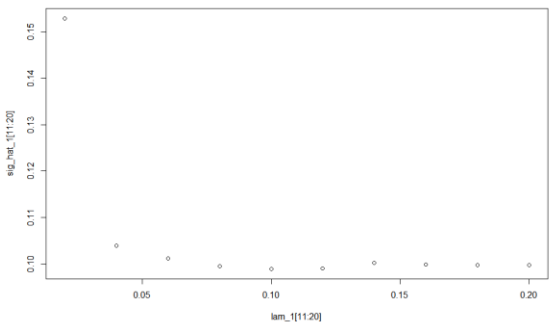
```
>
> param_opt = results[which.min(results[,"sse"]),]
> param_opt
```

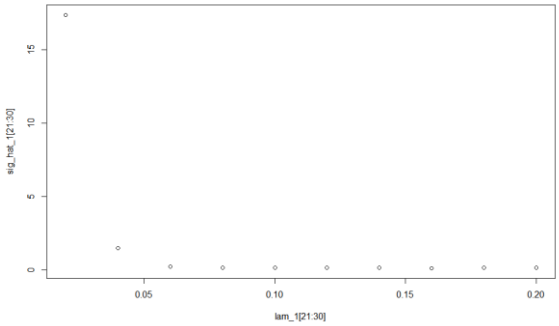
	model	lambda	sse	r2
14	1	0.3	7.796821	0.6796743

The best (least SSE) model was obtained for a model with degree = 1, and span = 0.3.

b. In order to calculate Cp, we started out by calculating residual standard error (sigma_hat) for a low bias model. The residual standard error was calculated for various values of lambda for each model (degree = 0,1, and 2). The graphs for these are shown below. As can be seen, for degree=0, sigma_hat values decrease slightly, but then continue to increase over the range. Hence, sigma_hat cannot be selected for this model.
   For degree=1, sigma_hat value of 0.10117629 (for lambda = 0.06) was chosen, while for degree = 2, sigma_hat value of 0.21362294 (for lambda = 0.06) was chosen.







The Cp was calculated for various models, and is shown in the table below:

Degree	Lambda	Cp
1	0.01	0.022406
1	0.02	0.015113
1	0.03	0.013138
1	0.04	0.012172
1	0.05	0.011623
1	0.1	0.01052
1	**0.3**	**0.009892**
1	0.5	0.009929
1	0.7	0.010065
1	0.9	0.010506
2	0.01	6.76E+12
2	0.02	183.717
2	0.03	0.801429
2	0.04	1.363419
2	0.05	0.165273
2	0.1	0.033956
2	0.6	0.014153
2	1.1	0.012031
2	1.6	0.011975
2	2.1	0.01194
2	2.6	0.011897

As can be seen above, the lowest Cp was obtained for model 1 (degree = 1) and span = 0.3. This agrees with what CV claims to be the best span and degree.

c. The CV prediction error standard deviation was calculated by using two approaches: taking the square root of the CV SSE/n (approach 1), or taking the standard deviation of the prediction error (difference between observed and predicted response in CV) (approach 2).

Approach 1:

```
> mean(sqrt(results$sse/n))
```

```
[1] 0.09953731
```

Approach 2:

```
> mean(sderror$sd)
[1] 0.09930618
```

d. To find the predicted cost, the predictors were first standardized.

The predicted cost for this model is $2515.02 (in dollar scale) or 3.40 (in log scale).

```
> scale_y(y.pred, mydata_orig_log$cost)
       1
3.400542
> 10^(scale_y(y.pred, mydata_orig_log$cost))
       1
2515.021
```

Problem 4

a. 10- fold cross validation with 20 CV replicates were conducted for nterms = 1 to 10 for PPR

```
> results
   model      sse       r2        sd
1      1 7.721126 0.6809310 0.09904902
2      2 7.768016 0.6790032 0.09934616
3      3 7.876023 0.6745583 0.10003106
4      4 8.063548 0.6668053 0.10121150
5      5 8.232073 0.6598414 0.10226580
6      6 8.362664 0.6544550 0.10307017
7      7 8.531825 0.6474661 0.10410257
8      8 8.701921 0.6404512 0.10513278
9      9 8.820394 0.6355538 0.10584761
10    10 8.983423 0.6288239 0.10681858
```

```
> param_opt
  model      sse      r2        sd
1     1 7.721126 0.680931 0.09904902
```

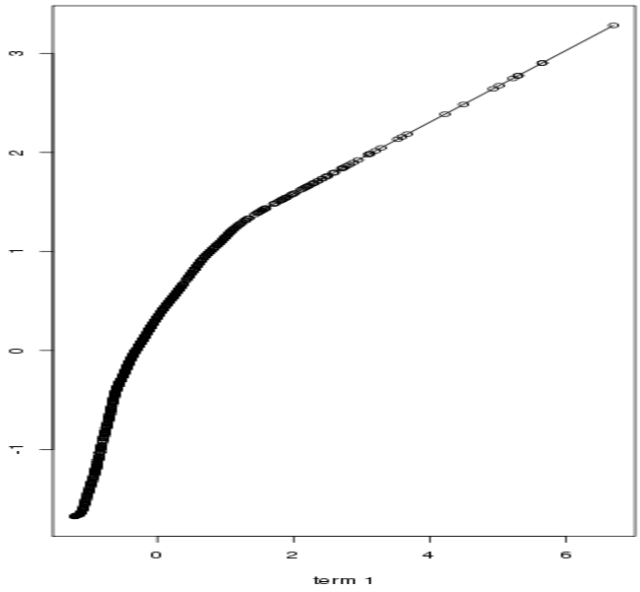
Best model for KNN: **nterms= 1** and CV R2 = 0.68

b. The prediction error standard deviation was calculated by using two approaches: taking the square root of the CV SSE/n, or taking the standard deviation of the prediction error (difference between observed and predicted response in CV).

```
> mean(sqrt(param_opt[,"sse"]/n))
[1] 0.09898678
>
> # or
>
> param_opt[,"sd"]
[1] 0.09904902
```

The best model with nterm = 1 was fitted. The results show that the largest coefficients are for drugs and comordibities. The plot suggests that as these variables increase together, the total cost increases rapidly and at after a certain point the increase becomes linear.

```
> ## fit best model
> out = ppr(total_cost~., data=mydata, nterms=ntermsBest)
> summary(out)
Call:
ppr(formula = total_cost ~ ., data = mydata, nterms = ntermsBest)

Goodness of fit:
 1 terms
7.425113

Projection direction vectors:
         age       gender interventions        drugs     ER_visits complications
 -0.02759078  -0.02686806    0.93275393  -0.07100287    0.10038411
0.15520531
comorbidities      duration
   0.28667863    0.08423339

Coefficients of ridge terms:
   term 1
0.1458987
```



c. To find the predicted cost, the values of the predictors for the new case was first standardized and then PPR model from part a. was run to find the predicted value.

The predicted cost is: $ 3240.684 (dollar scale), 3.510637 (log scale)

Problem 5

a. 10- fold cross validation with 20 CV replicates were conducted for K=1 to 10 for nearest neighbor models.

Kneighbors	misclass
1	0.1733645
2	0.1684579
3	0.15
4	0.1663551
5	0.1581776
6	0.1584112
7	0.1523364
8	0.1600467
9	0.1600467
10	0.1621495

Best model for KNN: **K = 3** and CV misclassification rate =0 .15

b. 3-fold cross validation with 10 CV replicates for GAM models with the 7 different set of predictors, which were found be sequentially removing the least significant variables until all variables in the model were significant.

```
[[1]]
type_bin ~ s(RI) + s(Na) + s(Mg) + s(Al) + s(Si) + s(K) + s(Ca) +  s(Ba) + s(Fe)
```

[[2]]
type_bin ~ s(RI) + s(Na) + s(Mg) + s(Al) + s(Si) + s(K) + s(Ba) + s(Fe)
[[3]]
type_bin ~ s(RI) + s(Na) + s(Mg) + s(Si) + s(K) + s(Ba) + s(Fe)
[[4]]
type_bin ~ s(RI) + s(Na) + s(Mg) + s(Si) + s(K) + s(Fe)
[[5]]
type_bin ~ s(RI) + s(Mg) + s(Si) + s(K) + s(Fe)
[[6]]
type_bin ~ s(RI) + s(Mg) + s(Si) + s(Fe)
[[7]]
type_bin ~ s(RI) + s(Mg) + s(Si)

func	misclass
1	0.20794393
2	0.19158879
3	0.18785047
4	0.19252336
5	0.18037383
6	0.15233645
7	0.16028037

Best model for GAMs is **"function 6"** with CV misclassification rate = 0.15233.

c) 3-fold cross validation with 20 CV replicates were conducted for nodes=1 to 15, and lambda = 0.1 to 2 in steps of 0.1.

nodes	lambda	misclass
1	0.1	0.1722
1	0.2	0.16729
1	0.3	0.16332
1	0.4	0.16285
1	0.5	0.16238
1	0.6	0.16145
1	0.7	0.16262
1	0.8	0.16215
1	0.9	0.16332
1	1	0.16098
1	1.1	0.16379

……..

15	1.2	0.15771
15	1.3	0.15864
15	1.4	0.15771
15	1.5	0.15724
15	1.6	0.15794
15	1.7	0.15771
15	1.8	0.15724
15	1.9	0.15794
15	2	0.15771

Best model for neural networks has parametres **nodes = 9 and lambda = 0.4** with CV misclassification rate = 0.148598

The best KNN, GAM and Neural Network models found in the previous sections were run using 10-fold cross validation with 50 replicates, resulting in the average CV misclassification rates of :

```
    KNN       GAM      NNET
0.1553271 0.1423364 0.1471028
```

This suggests that GAM and NNET perform slightly better than KNN, with GAM being overall the best model with lowest CV misclassification.

Problem 6

a. The best boosted tree model was chosen by looking at the error standard deviation obtained by using various values for interaction depth and shrinkage parameters. Using laplace loss function gave repeatedly higher error standard deviation than using gaussian loss function. Furthermore, since robustness to outliers did not seem particularly necessary for this problem, a gaussian loss function was used in the final model. The error SD values for various combinations of tuning parameters (using a gaussian loss function) are shown in the table below:

depth	shrinkage	SD
6	0.005	0.462248
6	0.01	0.461758
6	0.03	0.460967
6	0.05	0.463586
6	0.07	0.460017
6	0.1	0.468381
6	0.5	0.475261
7	0.005	0.462101
7	0.01	0.457117
7	0.03	0.467392
7	0.05	0.461649
7	0.07	0.46149
7	0.1	0.46604
7	0.5	0.484238

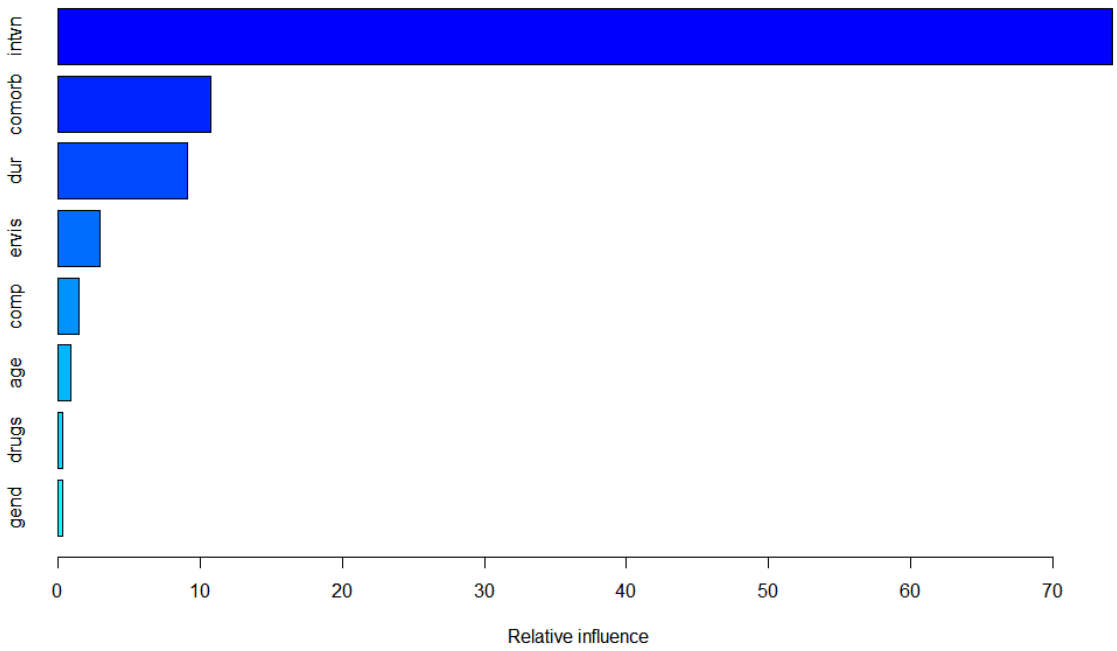
depth	shrinkage	SD
3	0.005	0.456951
3	0.01	0.463364
3	0.03	0.460264
3	0.05	0.456669
3	0.07	0.465616
3	0.1	0.463742
3	0.5	0.477219
4	0.005	0.461647
4	0.01	0.462124
4	0.03	0.461025
4	0.05	0.46379
4	0.07	0.460581
4	0.1	0.46119
4	0.5	0.483009
5	0.005	0.460373
5	0.01	0.466201
5	0.03	0.459479
5	0.05	0.462784
5	0.07	0.458439
5	0.1	0.458732
5	0.5	0.489832

The lowest error standard deviation (**0.457**) was obtained by using **interaction.depth = 3, shrinkage = 0.05, and Gaussian loss function**. These parameters were used in the consequent
The above measures were obtained without any standardization of variables, as tree-based models do not require any kind of standardization. However, in order to make the CV error standard deviation comparable to other models (which required standardization), the same model was applied to standardized dataset, which gave us an error SD of about 0.0979.

```
> sqrt(gbm.out$cv.error[best.iter])
[1] 0.09793824
```

b. As can be seen in the figure below, intvn is by far the most important, followed by comorb, dur, and ervis. Age and comp seem somewhat important. Drugs and gender seem the least important as per this model. These results are similar to those obtained from other models. Intvn, comorb, and dur are consistently deemed important, while drugs and gender are consistently deemed least important. The exact ranking of other predictors tend to vary somewhat across models, but are largely similar.

```
> summary(gbm.out,n.trees=best.iter)  # based on the optimal number of trees
        var       rel.inf
intvn intvn     74.2077723
comorb comorb 10.7306334
dur     dur      9.1159931
ervis   ervis    2.9671811
comp   comp  1.4547748
age     age  0.8503857
drugs drugs  0.3426726
gend   gend  0.3305870
```



c. The predicted cost for a person with age=59, gend=0, intvn=10, drugs=0, ervis=3, comp=0, comorb=4, and dur=300 is shown below:

```
> predCost.gbm <- predict(gbm.out, newdata=x.new); predCost.gbm
Using 106 trees...
[1] 3.496087
> 10^predCost.gbm
[1] 3133.91
```

The predicted cost here is $3133.91.

d) For the ischemic heart data, the test R-squared of various different models is shown below in ascending order.

Model	R-squared
KNN	0.5902
Linear Regression	0.6247
Tree	0.6348
GAM	0.6696
Kernel	0.6797

PPR	0.6809
Neural Net	0.6844
Boosted Trees	0.6936

As can be seen above, KNN and Linear regression tend perform the worst, while Boosted Trees and Neural Net perform the best for the given data set. GAM, Kernel, and PPR also do a fairly good job predicting the cost for the ischemic heart disease dataset.

Appendix

```
#### Load data ################################################
# My PC
main = "C:/Users/Steven/Documents/Academics/3_Graduate School/2014-2015 ~ NU/"

# Aginity
#main = "\\\\nas1/labuser169"

course = "MSIA_420_Predictive_Analytics"
datafolder = "Data"
setwd(file.path(main,course, datafolder))

filename = "HW2_data.csv"
mydata = read.csv(filename,header = T)

#### Process data #############################################
library(yaImpute)

head(mydata)
mydata = mydata[-1]
head(mydata) # drop ID column

mydata_orig = mydata

# convert rsponse to log10 (change 0 to 1)
hist(mydata$total_cost)
mydata$total_cost[mydata$total_cost == 0] = 1
mydata$total_cost = log10(mydata$total_cost)
hist(mydata$total_cost)

mydata_orig_log = mydata

resp = names(mydata)[1]  # response variable
pred = names(mydata)[-1] # predictor variables

mydata[pred] = sapply(mydata[pred], function(x) (x-mean(x))/sd(x)) #standardize predictors
mydata[resp] = (mydata[resp]-min(mydata[resp]))/(max(mydata[resp])-min(mydata[resp]))
```

```
#### Problem 1 ################################################

#### a) choose best K ####
# Use n-fold CV to find the best K for predicting
# cost using K-NN. What are the pros and cons of using
# n-fold CV, versus say 10-fold CV, for nearest neighbors?

# PLarger K means less bias towards overestimating the true
# expected error (as training folds will be closer to the total dataset)
# but higher variance and higher running time (as you are getting closer
# to the limit case: Leave-One-Out CV)
# n-fold equally computational effort. No need to run replicates compared
# to 10-fold. Use more neighbohrs, but only one observation to test so
# higher variance in the prediction

#R commands for creating indices of partition for K-fold CV

CVInd = function(n,K) {  #n is sample size; K is number of parts; returns K-length list of indices
for each part
  m = floor(n/K)  #approximate size of each part
  r=n-m*K
  I=sample(n,n)  #random reordering of the indices
  Ind=list()  #will be list of indices for all K parts
  length(Ind)=K
  for (k in 1:K) {
    if (k <= r) kpart = ((m+1)*(k-1)+1):((m+1)*k)
    else kpart=((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
    Ind[[k]] = I[kpart]  #indices for kth part of data
  }
  Ind
}

# CV to choose the best k

# Start the clock!
ptm <- proc.time()

Nrep = 1  #number of replicates of CV (if doing n-fold, need only 1 replicate)
n=nrow(mydata)
K = n  #K-fold CV on each replicate (n-fold)
y = mydata[,resp]
KNN_max = 10 # 1:KNN_max models
```

```
SSE = matrix(0,Nrep,KNN_max) # SSE for each model and each replicate
sderror =matrix(0,Nrep,KNN_max)

# repeat CV  Nrep replicates
for (j in 1:Nrep) {
  Ind = CVInd(n,K) # generate CV partition indices

  # repeat CV for each model using the same CV partition
  for (KNN in 1:KNN_max){

    yhat = y; # reset yhat (actually don't need this)

    # CV for each model
    for (k in 1:K) {
      train = as.matrix(mydata[-Ind[[k]],pred])
      test = as.matrix(mydata[Ind[[k]],pred])
      ytrain = mydata[-Ind[[k]],resp]

      out = ann(train,test,KNN,verbose=F)

      # when n-fold, then test has one observation. indices of out$Knn.. are in one row
      # and become a vector when subset. As.matrix converts this to matrix with n columns
      # equal to number of knn models. So need to transpose or add ncol = KNN and use matrix
function.
      ind = matrix(out$knnIndexDist[,1:KNN],ncol=KNN) # nearest neighbor indices for each obs
in test

      yhat[Ind[[k]]] = apply(ind,1,function(x) mean(ytrain[x])) # prediction = average of y of
neighbors

    } #end of k loop

    SSE[j,KNN] = sum((y-yhat)^2) # store SSE for this model for this CV replicate
    sderror[j,KNN] = sd(y-yhat)

  }# end of KNN loop

} #end of j loop

# Stop the clock
proc.time() - ptm

SSE
SSEAve = apply(SSE,2,mean);SSEAve
```

```
sderrorAve = apply(sderror,2,mean);sderrorAve

R2 = 1-SSEAve/n/var(y)
R2

K_opt = which.max(R2)
K_opt

plot(R2)
plot(yhat,y)

write.csv(K_opt, file = "P1_best_knn.csv",row.names=FALSE)
write.csv(SSEAve, file = "P1_cv_knn.csv",row.names=FALSE)

#### b) error ####
# For the optimal K from part (a), what is the CV
# estimate of the prediction error standard deviation?

# prediction error standard deviation
#
https://books.google.com/books?id=BB8fc1nTo_4C&pg=PA183&lpg=PA183&dq=%22prediction
+error+standard+deviation%22&source=bl&ots=q0zS8nHJks&sig=PMgtUdTiJzwV3FebgjoX_L-
f7rI&hl=en&sa=X&ei=GCjzVOSvH4GRyATQ-
IDgDA&ved=0CCUQ6AEwAzgK#v=onepage&q=%22prediction%20error%20standard%20devia
tion%22&f=false

mean(sqrt(SSE[,K_opt]/n))

# or

mean(sderror[,K_opt])

#### c) predict new case ####
# What is the predicted cost for a person with age=59,
# gend=0, intvn=10, drugs=0, ervis=3, comp=0, comorb=4, and dur=300?

train = as.matrix(mydata[,pred])
test = matrix(c(59,0,10,0,3,0,4,300),nrow=1,ncol=length(pred))
colnames(test) = pred

# standardize
mean_pred = apply(mydata_orig[pred],2,mean)
sds_pred = apply(mydata_orig[pred],2,sd)
test = apply(test,1,function(x) (x-mean_pred)/sds_pred)
```

```
test = t(test) # transpose to get obs in rows

out = ann(train,test,K_opt)
ind = matrix(out$knnIndexDist[,1:K_opt],ncol=K_opt)

# approach 1:
yorig = mydata_orig[,resp] # or use ytrain and unstandardize and unlogged
fit  = apply(ind,1,function(x) mean(yorig[x]))
fit

fit_log  = apply(ind,1,function(x) mean(y[x]))+-min(mydata_orig_log[resp])
fit_log = fit_log*(max(mydata_orig_log[resp])-
min(mydata_orig[resp]))+min(mydata_orig_log[resp])
fit_log
10^fit_log
```

Problem 2

a) GAM fit

```
# Fit a GAM model without interactions, and construct
# plots of the component functions. Which predictors
# appear to be the most relevant for predicting cost?

library(mgcv)  #stands for "Mixed GAM Computation Vehicle"

# reduce the degrees of freedom if fewer distinct values than default (10)
#s(gender) # 2
#s(drugs)  # 9
#s(complications # 3

# http://www.talkstats.com/showthread.php/39182-GAM-%28generalised-additive-models%29-
function
# http://r.789695.n4.nabble.com/gam-error-td2241518.html
# https://stat.ethz.ch/pipermail/r-help/2007-October/143569.html
# https://stat.ethz.ch/pipermail/r-help/2011-November/295047.html


# discrete variables to linear terms

out=gam(total_cost~s(age)+ gender + s(interventions)+ s(drugs,k=8) +  s(ER_visits) +
          complications +
          s(comorbidities) + s(duration),
        data=mydata, family=gaussian(), sp=c(-1,-1,-1,-1,-1,-1))

summary(out)
out$sp  ##estimated smoothing parameters for each constituent function
yhat=predict(out)
plot(yhat,mydata[,resp])  #probably quite a bit of overitting
##
par(mfrow=c(2,4))
plot(out)  #plot component functions
par(mfrow=c(1,1))

# intvn, comorb, and dur, comp, ervis seem
# The overall average effect (main effect, assuming not interactions) of the
# predictors on the response:
# As intervention increaes, the cost increases the most and then plateaus
# As comp and comrob increases, the costs slightly increases and then plateaus
```

```
# At high values of ervis, as ervis increae the cost increases at increasing rate
#
# The higher the edf , the higher complexity of smoothing function
# p-value shows age and drugs are not significant
# Signficance in decreasing order of signifcance based on pvalues:
# Most signifcant to least: como, interv, duration, complications, ervisits

#### b) GAM CV ####

# For the model from part (a), what is the CV estimate
# of the prediction error standard deviation? What are
# the pros and cons of using n-fold CV, versus say 10-fold CV, for GAMs?

#R commands for creating indices of partition for K-fold CV

CVInd = function(n,K) {  #n is sample size; K is number of parts; returns K-length list of indices
for each part
  m = floor(n/K)  #approximate size of each part
  r=n-m*K
  I=sample(n,n)  #random reordering of the indices
  Ind=list()  #will be list of indices for all K parts
  length(Ind)=K
  for (k in 1:K) {
    if (k <= r) kpart = ((m+1)*(k-1)+1):((m+1)*k)
    else kpart=((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
    Ind[[k]] = I[kpart]  #indices for kth part of data
  }
  Ind
}

# CV to choose the best k

# Start the clock!
ptm <- proc.time()

Nrep = 1  #number of replicates of CV (if doing n-fold, need only 1 replicate)
n=nrow(mydata)
K = n  #K-fold CV on each replicate (n-fold)
y = mydata[,resp]
models_max = 1 # number of maximum models to test

SSE = matrix(0,Nrep,models_max) # SSE for each model and each replicate
sderror =matrix(0,Nrep,models_max)
```

```
# repeat CV  Nrep replicates
for (j in 1:Nrep) {
  Ind = CVInd(n,K) # generate CV partition indices

  # repeat CV for each model using the same CV partition
  for (m in 1:models_max){

    yhat = y; # reset yhat (actually don't need this)

    # CV for each model
    for (k in 1:K) {
      train = mydata[-Ind[[k]],c(resp,pred)]
      test = mydata[Ind[[k]],pred]
      #ytrain = mydata[-Ind[[k]],resp]

      out=gam(total_cost~ gender + s(age)+s(interventions)+ s(drugs,k=8) +
                s(ER_visits)  + complications +s(comorbidities) + s(duration),
              data=train,
              family=gaussian(), sp=c(-1,-1,-1,-1,-1,-1))

      yhat[Ind[[k]]]=predict(out,newdata=test)

    } #end of k loop

    SSE[j,m] = sum((y-yhat)^2) # store SSE for this model for this CV replicate
    sderror[j,m] = sd(y-yhat)

  }# end of m loop

} #end of j loop

# Stop the clock
proc.time() - ptm

SSE
SSEAve = apply(SSE,2,mean);SSEAve
sderrorAve = apply(sderror,2,mean);sderrorAve

mean(sqrt(SSE[,1]/n))

# or

mean(sderror[,1])
```

```
#### c) predict new case ####
# What is the predicted cost for a person with age=59,
# gend=0, intvn=10, drugs=0, ervis=3, comp=0, comorb=4, and dur=300?

train = as.matrix(mydata[,c(resp,pred)])
test = matrix(c(59,0,10,0,3,0,4,300),nrow=1,ncol=length(pred))
colnames(test) = pred

# standardize
mean_pred = apply(mydata_orig[pred],2,mean)
sds_pred = apply(mydata_orig[pred],2,sd)
test = apply(test,1,function(x) (x-mean_pred)/sds_pred)
test = t(test) # transpose to get obs in rows

out=gam(total_cost~s(age)+ gender + s(interventions)+ s(drugs,k=8) +  s(ER_visits) +
complications +
         s(comorbidities) + s(duration),
       data=data.frame(train), family=gaussian(), sp=c(-1,-1,-1,-1,-1,-1))

yhat =predict(out,newdata=data.frame(test))
fit = yhat*(max(mydata_orig_log[resp])-
min(mydata_orig_log[resp]))+min(mydata_orig_log[resp])
fit
fit = 10^fit
fit


# Note this is not the same model as in CV, it includes complications, which had to be removed
# in CV because distinct values for complications becomes equal to 2 in some partitions.
```

Problem 3

```
head(mydata)
mydata = mydata[-1]
head(mydata) # drop ID column

mydata_orig = mydata

# convert rsponse to log10 (change 0 to 1)
hist(mydata$cost)
mydata$cost[mydata$cost == 0] = 1
mydata$cost = log10(mydata$cost)
hist(mydata$cost)

mydata_orig_log = mydata

resp = names(mydata)[1]  # response variable
pred = names(mydata)[-1] # predictor variables

mydata[pred] = sapply(mydata[pred], function(x) (x-mean(x))/sd(x)) #standardize predictors
mydata[resp] = (mydata[resp]-min(mydata[resp]))/(max(mydata[resp])-min(mydata[resp]))

#### a) CV best kernel

# Use CV to find the best combination of span and degree (0 for local average, 1 for local linear,
and 2 # for local quadratic regression) for a kernel method.

pred2 = pred[c(3,5,7,8)] # loes can take only 4 predictors

## Using CV to Compare Models for the mydata Data
#R commands for creating indices of partition for K-fold CV

CVInd = function(n,K) {  #n is sample size; K is number of parts; returns K-length list of indices
for each part
  m = floor(n/K)  #approximate size of each part
  r=n-m*K
  I=sample(n,n)  #random reordering of the indices
  Ind=list()  #will be list of indices for all K parts
  length(Ind)=K
  for (k in 1:K) {
    if (k <= r) kpart = ((m+1)*(k-1)+1):((m+1)*k)
    else kpart=((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
```

```
    Ind[[k]] = l[kpart]  #indices for kth part of data
  }
  Ind
}


## CV to choose the best k

# Start the clock!
ptm <- proc.time()

Nrep = 20  #number of replicates of CV (if doing n-fold, need only 1 replicate)
n=nrow(mydata)
K = 10  #K-fold CV on each replicate (n-fold)
y = mydata[,resp]

lambdas = c(0.01,0.03,0.05,0.1,0.3,0.5,0.7,0.9)
models = c(0,1,2)

#n iterations
Nrep*length(lambdas)*length(models)*K
loops = Nrep*length(lambdas)*length(models) # without folds

# SSE, R2 for each rep, model and lambda
results = c()
sderror = data.frame(numeric(0))
# repeat CV  Nrep replicates
for (j in 1:Nrep) {
  Ind = CVInd(n,K) # generate CV partition indices

  # repeat CV for each model using the same CV partition
  for (m in models){

    yhat = y; # reset yhat (actually don't need this)

    for(l in lambdas){

      # CV for each model
      for (k in 1:K) {
        train = mydata[-Ind[[k]],c(resp,pred2)] # does not work if you convert to
        # matrix and then to data frame
        test = mydata[Ind[[k]],pred2]
```

```
        # allow extrapolation : control = loess.control(surface = "direct")
        # Surface determines whether the fitted surface is computed directly at all
        # points ("direct") or whether an interpolation method is used ("interpolate").
        # The latter, the default, is what most users should use unless special circumstances
warrant.
        out=loess(cost ~., train ,degree=m, span=l,  control = loess.control(surface = "direct"))
        yhat[Ind[[k]]] = predict(out, newdata = test)

      } #end of k loop

      loops = loops - 1
      print(paste("Remaining Iterations: ",loops))

      results = rbind(results,c(j,m,l,sum((y-yhat)^2),1-var(y-yhat)/var(y)))
      sderror = rbind(sderror, c(sd(y-yhat)))
    } # end of l loop
  }# end of m loop

} #end of j loop

# Stop the clock
proc.time() - ptm
results <- as.data.frame(results)
colnames(results)=c("rep","model","lambda","sse","r2")

############
## Prediction Error SD (Part c)
mean(sqrt(results$sse/n))

########
results2 = aggregate(cbind(sse,r2)~  lambda + model, results, mean )
results2 = results2[c(2,1,3,4)] # sort
results2

param_opt = results2[which.min(results[,"sse"]),]
param_opt

modelBest = as.numeric(param_opt["model"])
lambdaBest = as.numeric(param_opt["lambda"])


#### b) Cp best kernel
# Use Cp to find the best combination of span and degree
```

```
# (0 for local average, 1 for local linear, and 2 for local quadratic regression) for a kernel
method. # Is this in agreement with what CV said was the best span and degree?


## Use Cp to choose lambda ##

###  first find sigma_hat for a low-bias model  ###
lam_1=c()
sig_hat_1=c()
model_1 = c()
for (model in c(0,1,2))
{
  for (lambda in seq(.02,.2,.02))
  {
    out=loess(cost ~., mydata[, c(1,4,6,8,9)],degree=model, span=lambda);
    print(c(lambda,out$s))
    model_1=rbind(model_1,model) #model vector
    lam_1=rbind(lam_1,lambda) #lambdas' vector
    sig_hat_1=rbind(sig_hat_1,out$s) #sig_hats' vector
  }
}

result = cbind(model_1,lam_1,sig_hat_1)
result
plot(lam_1[1:10],sig_hat_1[1:10]) #plot sig_hat vs lambda for model= 0
plot(lam_1[11:20],sig_hat_1[11:20]) #plot sig_hat vs lambda for model= 1
identify(lam_1[11:20],sig_hat_1[11:20]) # sig_hat for model = 1 is 0.10117629
plot(lam_1[21:30],sig_hat_1[21:30])  #plot sig_hat vs lambda for model= 2
identify(lam_1[21:30],sig_hat_1[21:30]) #sig_hat for model = 1 is 0.21362294

##now find Cp for various lambda###
sig_hat_1=0.10117629
sig_hat_2 = 0.21362294

#for model =1#
for (lambda in c(seq(.01,.05,.01), seq(.1,1,.2)))
{
  out=loess(cost ~., mydata[, c(1,4,6,8,9)],degree=1, span=lambda, control =
loess.control(surface = "direct"));
  SSE=sum((mydata[,1]-out$fitted)^2);
  Cp = (SSE+2*out$trace.hat*sig_hat_1^2)/nrow(mydata);
  print(c(lambda,Cp))
}
#smallest Cp = 0.01006446, lambda=0.50000000
```

```
#for model = 2#
for (lambda in c(seq(.01,.05,.01), seq(.1,3,.5)))
{
  out=loess(cost ~., mydata[, c(4,6,8,9,1)],degree=2, span=lambda);
  SSE=sum((mydata[,1]-out$fitted)^2);
  Cp = (SSE+2*out$trace.hat*sig_hat_2^2)/nrow(mydata);
  print(c(lambda,Cp))
}
#smallest Cp = 0.01336518 lambda=0.90000000

##Use Cp to choose the combination: model = 1 and lambda = 0.04

############### (d) ##############

out<-loess(cost ~., mydata[, c(1,4,6,8,9)],degree=1, span=0.3, control = loess.control(surface =
"direct"));

x.new = as.data.frame(t(c(59,0,10,0,3,0,4,300)))
data.means <- sapply(mydata_orig[2:9],mean)
data.sd <- sapply(mydata_orig[2:9],sd)
x.new.2 <- as.data.frame((x.new-data.means)/data.sd)


names(x.new.2) <- names(mydata)[2:9]

## Predicted value in log scale
y.pred = predict(out,newdata=x.new.2);y.pred

## Predicted value in dollar scale
10^(scale_y(y.pred, mydata_orig_log$cost))
```

Problem 4

a) PPR CV

```
# For the model from part (a), what is the CV estimate
# of the prediction error standard deviation? What are
# the pros and cons of using n-fold CV, versus say 10-fold CV, for GAMs?

#R commands for creating indices of partition for K-fold CV

CVInd = function(n,K) {  #n is sample size; K is number of parts; returns K-length list of indices
for each part
  m = floor(n/K)  #approximate size of each part
  r=n-m*K
  I=sample(n,n)  #random reordering of the indices
  Ind=list()  #will be list of indices for all K parts
  length(Ind)=K
  for (k in 1:K) {
          if (k <= r) kpart = ((m+1)*(k-1)+1):((m+1)*k)
          else kpart=((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
          Ind[[k]] = I[kpart]  #indices for kth part of data
  }
  Ind
}

# CV to choose the best k

# Start the clock!
ptm <- proc.time()

Nrep = 20  #number of replicates of CV (if doing n-fold, need only 1 replicate)
n=nrow(mydata)
K = 10  #K-fold CV on each replicate (n-fold)
y = mydata[,resp]
models_max = 10 # number of maximum models to test

#SSE = matrix(0,Nrep,models_max) # SSE for each model and each replicate
#sderror =matrix(0,Nrep,models_max)

#n iterations , 1.36 per iteraion
Nrep*models_max*K
loops = Nrep*models_max # without folds
```

```
# SSE, R2 for each rep, model and lambda
results = c()


# repeat CV  Nrep replicates
for (j in 1:Nrep) {
  Ind = CVInd(n,K) # generate CV partition indices

  # repeat CV for each model using the same CV partition
  for (m in 1:models_max){

        yhat = y; # reset yhat (actually don't need this)

        # CV for each model
        for (k in 1:K) {
        train = mydata[-Ind[[k]],c(resp,pred)]
        test = mydata[Ind[[k]],pred]
        #ytrain = mydata[-Ind[[k]],resp]

        out = ppr(total_cost~., data=train, nterms=m)

        yhat[Ind[[k]]]=predict(out,newdata=test)

        } #end of k loop

        #SSE[j,m] = sum((y-yhat)^2) # store SSE for this model for this CV replicate
        #sderror[j,m] = sd(y-yhat)

        results = rbind(results,c(j,m,sum((y-yhat)^2),1-var(y-yhat)/var(y),sd(y-yhat)))

        loops = loops - 1
        print(paste("Remaining Iterations: ",loops))

  }# end of m loop

} #end of j loop

# Stop the clock
proc.time() - ptm

colnames(results)=c("rep","model","sse","r2","sd")
results = aggregate(cbind(sse,r2,sd)~  model, results, mean )
#results = results[c(2,1,3,4)] # sort
results
```

```
param_opt = results[which.min(results[,"sse"]),]
param_opt

modelBest = as.numeric(param_opt["model"])
ntermsBest = modelBest

write.csv(param_opt, file = "P4_best_ppr.csv",row.names=FALSE)
write.csv(results, file = "P4_results_ppr.csv",row.names=FALSE)

#### b) Error + Model + Plots

mean(sqrt(param_opt[,"sse"]/n))

# or

param_opt[,"sd"]

## fit best model
out = ppr(total_cost~., data=mydata, nterms=ntermsBest)
summary(out)
par(mfrow=c(1,3))
plot(out)  #plot component functions
par(mfrow=c(1,1))


#### c) predict new case ####
# What is the predicted cost for a person with age=59,
# gend=0, intvn=10, drugs=0, ervis=3, comp=0, comorb=4, and dur=300?

train = as.matrix(mydata[,c(resp,pred)])
test = matrix(c(59,0,10,0,3,0,4,300),nrow=1,ncol=length(pred))
colnames(test) = pred

# standardize
mean_pred = apply(mydata_orig[pred],2,mean)
sds_pred = apply(mydata_orig[pred],2,sd)
test = apply(test,1,function(x) (x-mean_pred)/sds_pred)
test = t(test) # transpose to get obs in rows

yhat =predict(out,newdata=data.frame(test))

fit = yhat*(max(mydata_orig_log[resp])-
min(mydata_orig_log[resp]))+min(mydata_orig_log[resp])
```

```
fit
fit = 10^fit
fit
```

Problem 5

```
library(yaImpute)
mydata=read.table("fgl.txt",sep="\t")
mydata_orig = mydata

z=(mydata$type == "WinF") | (mydata$type == "WinNF")
y=as.character(mydata$type)
y[z]="Win"; y[!z]="Other"
mydata=data.frame(mydata,"type_bin"=as.factor(y))  #add a binary factor response column
y[y == "Win"]=1;y[y == "Other"]=0;
mydata=data.frame(mydata,"type01"=as.numeric(y))  #also add a binary numeric response
column

pred = names(mydata)[1:9]
resp = names(mydata)[11]

mydata[pred]=sapply(mydata[pred], function(x) (x-mean(x))/sd(x)) #standardize predictors
train=as.matrix(mydata[,pred]); test=as.matrix(mydata[,pred])
ytrain=mydata[,resp]; ytest=mydata[,resp]

#### Part a: KNN ####
K=5
out=ann(train,test,K)
ind=as.matrix(out$knnIndexDist[,1:K],ncol=K)
phat=apply(ind,1,function(x) sum(ytrain[x]=="Win")/length(ytrain[x]))
result = phat
result[phat>=0.5] = "Win"
result[phat<0.5] = "Other"
result = as.factor(result)
result

# plot(phat,jitter(as.numeric(ytest=="Win"),amount=.05))

####can alternatively use the following
library(class)

out=knn(train, test, ytrain, k = 5, prob = F)
out
table(out,result)
sum(out!=result)/length(out)

tab = table(ytest,out)
```

```
prop.table(tab,1)
prop.table(tab)
1-sum(diag(prop.table(tab))) # misclassification
sum(ytest!=result)/length(y)

#R commands for creating indices of partition for K-fold CV

CVInd = function(n,K) {  #n is sample size; K is number of parts; returns K-length list of indices
for each part
  m = floor(n/K)  #approximate size of each part
  r=n-m*K
  I=sample(n,n)  #random reordering of the indices
  Ind=list()  #will be list of indices for all K parts
  length(Ind)=K
  for (k in 1:K) {
          if (k <= r) kpart = ((m+1)*(k-1)+1):((m+1)*k)
          else kpart=((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
          Ind[[k]] = I[kpart]  #indices for kth part of data
  }
  Ind
}

# CV to choose the best k

# Start the clock!
ptm <- proc.time()

Nrep = 1 #number of replicates of CV (if doing n-fold, need only 1 replicate)
n=nrow(mydata)
K = 3  #K-fold CV on each replicate (n-fold)
y = mydata[,resp]

models = seq(1,10,1)

#n iterations , 1.36 per iteraion
Nrep*length(models)*K

#misclass= matrix(0,Nrep*length(lambdas)*length(models),4) # misclass for each model and
each replicate
misclass = c()

# repeat CV  Nrep replicates
for (j in 1:Nrep) {
  Ind = CVInd(n,K) # generate CV partition indices
```

```
  # repeat CV for each model using the same CV partition
  for (m in models){

        yhat = y; # reset yhat (actually don't need this)

        # CV for each model
        for (k in 1:K) {
        train = as.matrix(mydata[-Ind[[k]],pred])
        test = as.matrix(mydata[Ind[[k]],pred])
        ytrain = mydata[-Ind[[k]],resp]

        # out = ann(train,test,KNN,verbose=F)

        # when n-fold, then test has one observation. indices of out$Knn.. are in one row
        # and become a vector when subset. As.matrix converts this to matrix with n columns
        # equal to number of knn models. So need to transpose or add ncol = KNN and use
matrix function.
        # ind = matrix(out$knnIndexDist[,1:KNN],ncol=KNN) # nearest neighbor indices for each
obs in test

        # phat=apply(ind,1,function(x) sum(ytrain[x]=="Win")/length(ytrain[x]))
        # result = phat
        # result[phat>=0.5] = "Win"
        # result[phat<0.5] = "Other"
        # result = as.factor(result)

        out=knn(train, test, ytrain, m, prob = F); result = out
        yhat[Ind[[k]]] = result # prediction = average of y of neighbors

        } #end of k loop
        #tab = table(y,yhat)
        #miss = 1-sum(diag(prop.table(tab)))
        #misclass[j,KNN] = miss
        misclass = rbind(misclass,c(j,m,sum(y!=yhat)/length(y)))


  }# end of KNN loop

} #end of j loop

# Stop the clock
proc.time() - ptm
```

```
colnames(misclass) = c("rep","Kneighbors","misclass")
misclass
misclassAVE = aggregate(misclass~ Kneighbors, misclass, mean ) # aggregate reps
misclassAVE = misclassAVE[order(misclassAVE[,"Kneighbors"]),] # sort
misclassAVE

param_opt = misclassAVE[which.min(misclassAVE[,"misclass"]),]
param_opt

write.csv(param_opt, file = "P5_best_knn.csv",row.names=FALSE)

par(mfrow=c(1,1))
plot(misclassAVE)

write.csv(misclassAVE, file = "P5_misclassAVE_knn.csv",row.names=FALSE)


#### Part b: GAM ####

mydata=read.table("fgl.txt",sep="\t")
mydata_orig = mydata

z=(mydata$type == "WinF") | (mydata$type == "WinNF")
y=as.character(mydata$type)
y[z]="Win"; y[!z]="Other"
mydata=data.frame(mydata,"type_bin"=as.factor(y))  #add a binary factor response column
y[y == "Win"]=1;y[y == "Other"]=0;
mydata=data.frame(mydata,"type01"=as.numeric(y))  #also add a binary numeric response
column

pred = names(mydata)[1:9]
resp = names(mydata)[11]

mydata[pred]=sapply(mydata[pred], function(x) (x-mean(x))/sd(x)) #standardize predictors
train=as.matrix(mydata[,pred]); test=as.matrix(mydata[,pred])
ytrain=mydata[,resp]; ytest=mydata[,resp]

library(mgcv)  #stands for "Mixed GAM Computation Vehicle"

# reduce the degrees of freedom if fewer distinct values than default (10)
#s(gender) # 2
#s(drugs)  # 9
#s(complications # 3
```

```
# http://www.talkstats.com/showthread.php/39182-GAM-%28generalised-additive-models%29-function
# http://r.789695.n4.nabble.com/gam-error-td2241518.html
# https://stat.ethz.ch/pipermail/r-help/2007-October/143569.html
# https://stat.ethz.ch/pipermail/r-help/2011-November/295047.html

fList = list()

f= formula(type_bin~ s(RI)+s(Na)+s(Mg)+s(Al)+s(Si)+s(K)+s(Ca)+s(Ba)+s(Fe))
fList = append(fList,f)
npred = length(all.vars(f)[-1])
sp = rep(-1, npred)
out=gam(fList[[1]] , data=mydata, family=binomial(), sp=sp)

summary(out) # remove s(Ca)

f = update(f, . ~ . -s(Ca))
fList = append(fList,f)
npred = length(all.vars(f)[-1])
sp = rep(-1, npred)
out=gam(f ,data=mydata, family=binomial(), sp=sp)

summary(out) # remove s(Al)

f = update(f, . ~ . -s(Al))
fList = append(fList,f)
npred = length(all.vars(f)[-1])
sp = rep(-1, npred)
out=gam(f ,data=mydata, family=binomial(), sp=sp)

summary(out) # remove s(Ba)

f = update(f, . ~ . -s(Ba))
fList = append(fList,f)
npred = length(all.vars(f)[-1])
sp = rep(-1, npred)
out=gam(f ,data=mydata, family=binomial(), sp=sp)

summary(out) # remove s(Na)

f = update(f, . ~ . -s(Na))
fList = append(fList,f)
npred = length(all.vars(f)[-1])
```

```
sp = rep(-1, npred)
out=gam(f ,data=mydata, family=binomial(), sp=sp)

summary(out) # remove s(K)

f = update(f, . ~ . -s(K))
fList = append(fList,f)
npred = length(all.vars(f)[-1])
sp = rep(-1, npred)
out=gam(f ,data=mydata, family=binomial(), sp=sp)

summary(out) # remove s(Fe)

f = update(f, . ~ . -s(Fe))
fList = append(fList,f)
npred = length(all.vars(f)[-1])
sp = rep(-1, npred)
out=gam(f ,data=mydata, family=binomial(), sp=sp)

summary(out) # all sig

out=gam(fList[[7]] ,data=mydata, family=binomial(), sp=sp)

out$sp  ##estimated smoothing parameters for each constituent function
phat=predict(out, type="response")

yhat = phat
yhat[phat>=0.5] = "Win"
yhat[phat<0.5] = "Other"
yhat = as.factor(yhat)
yhat

y = mydata[,resp]
table(y,yhat)

tab = table(y,yhat)
prop.table(tab,1)
prop.table(tab)
1-sum(diag(prop.table(tab))) # misclassification
sum(y!=yhat)/length(y) # same

plot(yhat,mydata[,resp])  #probably quite a bit of overitting
##
par(mfrow=c(2,4))
```

```
plot(out)  #plot component functions
par(mfrow=c(1,1))

#R commands for creating indices of partition for K-fold CV

CVInd = function(n,K) {  #n is sample size; K is number of parts; returns K-length list of indices
for each part
  m = floor(n/K)  #approximate size of each part
  r=n-m*K
  I=sample(n,n)  #random reordering of the indices
  Ind=list()  #will be list of indices for all K parts
  length(Ind)=K
  for (k in 1:K) {
          if (k <= r) kpart = ((m+1)*(k-1)+1):((m+1)*k)
          else kpart=((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
          Ind[[k]] = I[kpart]  #indices for kth part of data
  }
  Ind
}

# CV to choose the best k

# Start the clock!
ptm <- proc.time()

Nrep = 20  #number of replicates of CV (if doing n-fold, need only 1 replicate)
n=nrow(mydata)
K = 3  #K-fold CV on each replicate (n-fold)
y = mydata[,resp]

fList = fList[1:7]
models = seq(1,length(fList),1)

#n iterations , 1.36 per iteraion
Nrep*length(models)*K

#misclass= matrix(0,Nrep*length(lambdas)*length(models),4) # misclass for each model and
each replicate
misclass = c()

# repeat CV  Nrep replicates
for (j in 1:Nrep) {
  Ind = CVInd(n,K) # generate CV partition indices
```

```
  # repeat CV for each model using the same CV partition
  for (m in models){

        yhat = y; # reset yhat (actually don't need this)
        npred = length(all.vars(fList[[m]])[-1]) # number of predictors
        sp = rep(-1, npred) # parameter for GAM

        # CV for each model
        for (k in 1:K) {
        train = mydata[-Ind[[k]],c(resp,pred)] # doesn't work with matrix input for fit function
        test = mydata[Ind[[k]],pred]
        #ytrain = mydata[-Ind[[k]],resp]

        out=gam(fList[[m]] ,data=train, family=binomial(), sp=sp)

        phat=predict(out,newdata=test, type="response")

        yhat2 = phat
        yhat2[phat>=0.5] = "Win"
        yhat2[phat<0.5] = "Other"
        yhat2 = as.factor(yhat2)

        yhat[Ind[[k]]]=yhat2

        } #end of k loop

        # tab = table(y,yhat)
        # miss = 1-sum(diag(prop.table(tab)))
        # misclass[j,m] = miss
        misclass = rbind(misclass,c(j,m,sum(y!=yhat)/length(y)))

  }# end of m loop

} #end of j loop

# Stop the clock
proc.time() - ptm

colnames(misclass) = c("rep","func","misclass")
misclass
misclassAVE = aggregate(misclass~ func, misclass, mean ) # aggregate reps
misclassAVE = misclassAVE[order(misclassAVE[,"func"]),] # sort
misclassAVE
```

```
param_opt = misclassAVE[which.min(misclassAVE[,"misclass"]),]
param_opt

write.csv(param_opt, file = "P5_best_gam.csv",row.names=FALSE)

par(mfrow=c(1,1))
plot(misclassAVE)

write.csv(misclassAVE, file = "P5_misclassAVE_gam.csv",row.names=FALSE)


#### Part c:  NNET ####


## Read data, convert response to binary, and standardize predictors

mydata = read.table("fgl.txt",sep="\t")
z = (mydata$type == "WinF") | (mydata$type == "WinNF")
y = as.character(mydata$type)
y[z] = "Win"; y[!z] = "Other"
mydata = data.frame(mydata,"type_bin"=as.factor(y))  #add a binary factor response column
y[y == "Win"] = 1;y[y == "Other"] = 0;
mydata = data.frame(mydata,"type01"=as.numeric(y))  #also add a binary numeric response
column

pred = names(mydata)[1:9]
resp = names(mydata)[11]

mydata[pred]=sapply(mydata[pred], function(x) (x-mean(x))/sd(x)) #standardize predictors
train=as.matrix(mydata[,pred]); test=as.matrix(mydata[,pred])
ytrain=mydata[,resp]; ytest=mydata[,resp]

## fit nnet

library(nnet)
mydata.nn1 = nnet(type_bin~., mydata[,c(pred,resp)],
                  linout=F, skip=F, size=10, decay=.05, maxit=1000, trace=F)

phat = as.numeric(predict(mydata.nn1))
y = mydata[[12]]
yhat = as.numeric(phat >= 0.5)  #classify as 1 if predicted probability >= 0.5
sum(y != yhat)/length(y)  #misclassification rate
summary(mydata.nn1)
plot(phat,jitter(y,0.05))
```

```
# alternative approach
y = mydata[,resp]
yhat = predict(mydata.nn1,type="class")
sum(y != yhat)/length(y)

## Using CV to Compare Models for the mydata Data
#R commands for creating indices of partition for K-fold CV

CVInd = function(n,K) {  #n is sample size; K is number of parts; returns K-length list of indices
for each part
  m = floor(n/K)  #approximate size of each part
  r=n-m*K
  I=sample(n,n)  #random reordering of the indices
  Ind=list()  #will be list of indices for all K parts
  length(Ind)=K
  for (k in 1:K) {
          if (k <= r) kpart = ((m+1)*(k-1)+1):((m+1)*k)
          else kpart=((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
          Ind[[k]] = I[kpart]  #indices for kth part of data
  }
  Ind
}

## CV to choose the best k

# Start the clock!
ptm <- proc.time()

Nrep = 20  #number of replicates of CV (if doing n-fold, need only 1 replicate)
n=nrow(mydata)
K = 3  #K-fold CV on each replicate (n-fold)
y = mydata[,resp]

models = seq(1,15,1)
lambdas = seq(.1,2,.1)

#n iterations , 1.36 per iteraion
Nrep*length(lambdas)*length(models)*K

#misclass= matrix(0,Nrep*length(lambdas)*length(models),4) # misclass for each model and
each replicate
misclass = c()
```

```
# repeat CV  Nrep replicates
for (j in 1:Nrep) {
  Ind = CVInd(n,K) # generate CV partition indices

  # repeat CV for each model using the same CV partition
  for (m in models){

        yhat = y; # reset yhat (actually don't need this)

        for(l in lambdas){

        # CV for each model
        for (k in 1:K) {
        train = mydata[-Ind[[k]],c(resp,pred)] # does not work if you convert to
        # matrix and then to data frame
        test = mydata[Ind[[k]],pred]
        #ytrain = mydata[-Ind[[k]],resp]

        out = nnet(type_bin~., data =train,
                linout=F, skip=F, size=m, decay=l, maxit=1000, trace=F)

        yhat[Ind[[k]]] = predict(out, newdata = test,type="class")

        } #end of k loop

        #tab = table(y,yhat)
        #miss = 1-sum(diag(prop.table(tab)))
        #misclass[j,KNN] = miss
        misclass = rbind(misclass,c(j,m,l,sum(y!=yhat)/length(y)))

        } # end of l loop
  }# end of m loop

} #end of j loop

# Stop the clock
proc.time() - ptm

colnames(misclass) = c("rep","nodes","lambda","misclass")
misclass
misclassAVE = aggregate(misclass~ nodes + lambda, misclass, mean ) # aggregate reps
misclassAVE = misclassAVE[order(misclassAVE[,"nodes"]),] # sort
misclassAVE
```

```
param_opt = misclassAVE[which.min(misclassAVE[,"misclass"]),]
param_opt

write.csv(param_opt, file = "P5_best_nnet.csv",row.names=FALSE)

par(mfrow=c(1,1))
plot(misclassAVE)

write.csv(misclassAVE, file = "P5_misclassAVE_nnet.csv",row.names=FALSE)

#### Part c: KNN,GAM vs NNET ####

CVInd = function(n,K) {  #n is sample size; K is number of parts; returns K-length list of indices
for each part
  m = floor(n/K)  #approximate size of each part
  r=n-m*K
  I=sample(n,n)  #random reordering of the indices
  Ind=list()  #will be list of indices for all K parts
  length(Ind)=K
  for (k in 1:K) {
          if (k <= r) kpart = ((m+1)*(k-1)+1):((m+1)*k)
          else kpart=((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
          Ind[[k]] = I[kpart]  #indices for kth part of data
  }
  Ind
}

# CV to choose the best k

# Start the clock!
ptm <- proc.time()

Nrep = 1  #number of replicates of CV (if doing n-fold, need only 1 replicate)
n=nrow(mydata)
K = 3  #K-fold CV on each replicate (n-fold)
y = mydata[,resp]

fBest = 6
kBest = 3
lambdaBest = 0.4
nodeBest = 9


#n iterations , 1.36 per iteraion
```

```
Nrep*K

#misclass= matrix(0,Nrep*length(lambdas)*length(models),4) # misclass for each model and
each replicate
misclass = c()

# repeat CV  Nrep replicates
for (j in 1:Nrep) {
  Ind = CVInd(n,K) # generate CV partition indices

  yhat_KNN = y; # reset yhat (actually don't need this)
  yhat_GAM = y; # reset yhat (actually don't need this)
  yhat_NNET = y; # reset yhat (actually don't need this)

  npred = length(all.vars(fList[[fBest]])[-1]) # number of predictors
  sp = rep(-1, npred) # parameter for GAM

  # CV for each model
  for (k in 1:K) {
        train = mydata[-Ind[[k]],c(resp,pred)] # doesn't work with matrix input for fit function
        test = mydata[Ind[[k]],pred]
        ytrain = mydata[-Ind[[k]],resp]

        # KNN (don't know why knn needs same number of cols in test and train)
        out=knn(train[,c(pred)], test, ytrain, kBest, prob = F); result = out
        yhat_KNN[Ind[[k]]] = result # prediction = average of y of neighbors


        # GAM
        out=gam(fList[[fBest]] ,data=train, family=binomial(), sp=sp)
        phat=predict(out,newdata=test, type="response")
        yhat2 = phat
        yhat2[phat>=0.5] = "Win"
        yhat2[phat<0.5] = "Other"
        yhat2 = as.factor(yhat2)
        yhat_GAM[Ind[[k]]]=yhat2

        # NNET
        out = nnet(type_bin~., data =train,
        linout=F, skip=F, size=nodeBest, decay=lambdaBest, maxit=1000, trace=F)

        yhat_NNET[Ind[[k]]] = predict(out, newdata = test,type="class")
```

```
  } #end of k loop

  # tab = table(y,yhat)
  # miss = 1-sum(diag(prop.table(tab)))
  # misclass[j,m] = miss

  misclass = rbind(misclass,c(j,sum(y!=yhat_KNN)/length(y),
                              sum(y!=yhat_GAM)/length(y),
                              sum(y!=yhat_NNET)/length(y)))


} #end of j loop

# Stop the clock
proc.time() - ptm

colnames(misclass) = c("rep","KNN","GAM","NNET")
misclass

misclassAVE = apply(misclass,2,mean)[-1]
misclassAVE

model_opt = which.min(misclassAVE)
model_opt

write.csv(model_opt, file = "P5_best.csv",row.names=FALSE)

par(mfrow=c(1,1))
plot(misclassAVE)

write.csv(misclassAVE, file = "P5_misclassAVE_best.csv",row.names=FALSE)
```

Problem 6

```
################
## Question 6 ##
################

# Load Data
setwd("C:\\Users\\Sanjeevni\\code\\msia420")
data <- read.csv("C:\\Users\\Sanjeevni\\Documents\\1 - Northwestern\\2015 Winter\\Predictive
Analytics\\HW2_data.csv")
data$y <- log10(data$cost)
data.gbm <- data[3:11]

# Standardized version. Used later for SSE comparison to other standardized models
data.gbm.std <- data.gbm
data.gbm.std[1:8] <- sapply(data.gbm.std[1:8], function(x) (x-mean(x))/sd(x))
data.gbm.std[9] <- (data.gbm.std[9]-min(data.gbm.std[9]))/(max(data.gbm.std[9])-
min(data.gbm.std[9]))


# Gaussian
CV_err_sd_gauss = data.frame(numeric(0), numeric(0), numeric(0))
library(gbm)
for(j in 3:7){
  for(s in c(.005,.01,.03,.05,.07,.1,.5)){
    gbm.out <- gbm(y~., data=data.gbm, var.monotone=rep(0,8), distribution="gaussian",
n.trees=5000, shrinkage=s, interaction.depth=j, bag.fraction = .5, train.fraction = 1,
n.minobsinnode = 10, cv.folds = 10, keep.data=TRUE, verbose=FALSE)
    best.iter <- gbm.perf(gbm.out,method="cv");best.iter
    CV_err_sd_gauss <- rbind(CV_err_sd_gauss,c(j,s,sqrt(gbm.out$cv.error[best.iter]))) #CV
error SD
  }
}
names(CV_err_sd_gauss) <- c("depth", "shrinkage", "SD")

# Laplace
CV_err_sd_lp_2 = data.frame(numeric(0), numeric(0), numeric(0))
for(j in 3:7){
  for(s in c(.005,.01,.03,.05,.07,.1,.5)){
    gbm.out <- gbm(y~., data=data.gbm, var.monotone=rep(0,8), distribution="laplace",
n.trees=5000, shrinkage=s, interaction.depth=j, bag.fraction = .5, train.fraction = 1,
n.minobsinnode = 10, cv.folds = 10, keep.data=TRUE, verbose=FALSE)
    best.iter <- gbm.perf(gbm.out,method="cv");best.iter
```

```
   CV_err_sd_lp_2 <- rbind(CV_err_sd_lp_2,c(j,s,sqrt(gbm.out$cv.error[best.iter]))) #CV error
SD
  }
}
names(CV_err_sd_lp_2) <- c("depth", "shrinkage", "SD")

# Final selected model based on best depth and shrinkage values selected above

gbm.out <- gbm(y~., data=data.gbm, var.monotone=rep(0,8), distribution="gaussian",
n.trees=5000, shrinkage=0.05, interaction.depth=3, bag.fraction = .5, train.fraction = 1,
n.minobsinnode = 10, cv.folds = 10, keep.data=TRUE, verbose=FALSE)
best.iter <- gbm.perf(gbm.out,method="cv");best.iter

err_sd = sqrt(gbm.out$cv.error[best.iter])
var_y =  (sd(data.gbm$y))^2
1-((err_sd^2)/(var_y)) # R-squared

##
par(mfrow=c(1,1))
summary(gbm.out,n.trees=best.iter)  # based on the optimal number of trees
##
par(mfrow=c(1,3))
for (i in 1:8) plot(gbm.out, i.var = i, n.trees = best.iter)
##
plot(gbm.out, i.var = c(2,3), n.trees = best.iter)
##
print(pretty.gbm.tree(gbm.out,1))  #show the first tree
##
print(pretty.gbm.tree(gbm.out,best.iter))  #show the last tree

###### c #######

## age=59, gend=0, intvn=10, drugs=0, ervis=3, comp=0, comorb=4, and dur=300
x.new = as.data.frame(t(c(59,0,10,0,3,0,4,300)))
names(x.new) <- names(data.gbm)[1:8]
# Predicted cost in log scale
predCost.gbm <- predict(gbm.out, newdata=x.new); predCost.gbm
# Predicted cost on dollar scale
predCost.gbm.dollar = 10^predCost.gbm
```