

PA2 - Homework 3

Kristiyan Dimitrov

05/03/2020

Loading required package: nlme

This is mgcv 1.8-28. For overview type 'help("mgcv-package")'.

Problem 1 - KNN

New names:

* `` -> ...1

	age	gend	intvn	drugs	ervis	comp
1	0.63394297	-0.5437618	-0.48382784	0.5200762	0.2179635	-0.2302054
2	0.04171172	-0.5437618	-0.48382784	-0.4198780	0.9762649	-0.2302054
3	0.48588516	-0.5437618	2.19729972	-0.4198780	-0.5403379	-0.2302054
4	0.18976953	1.8367066	0.76736502	-0.4198780	1.3554156	-0.2302054
5	-0.55051953	-0.5437618	0.05239767	1.4600303	1.3554156	-0.2302054
6	1.07811641	-0.5437618	-0.66256968	-0.4198780	-0.1611872	-0.2302054

	comorb	dur	log10_cost
1	-0.1288017	1.1244965	2.253096
2	-0.6329195	-0.3641411	2.503791
3	0.2072769	1.5628176	3.968982
4	-0.2968410	1.3891432	2.448552
5	-0.6329195	-1.2077024	4.272471
6	0.0392376	1.0914157	2.656482

(a) - Using n-fold CV to find the best K for K-NN

Using 20 repetitions of n-fold CV (aka LOOCV) to compare K=5 to K=10

```
[1] 0.2963158 0.2849334
```

```
[1] 0 0
```

```
[1] 0.5677138 0.5843193
```

The first row shows the average CV MSE for the two models. The second row shows us 0 standard deviation for both. This is expected behavior, because we are doing LOOCV. In other words, just like in HW1, the average MSE across all folds for a given randomized split of data will be the same across all splits, because we eventually go through each data point one at a time.

Looking at the first few rows of the MSE table, we confirm that the average MSE across different iterations of CV is the same.

```

      [,1]      [,2]
[1,] 0.2963158 0.2849334
[2,] 0.2963158 0.2849334
[3,] 0.2963158 0.2849334
[4,] 0.2963158 0.2849334
[5,] 0.2963158 0.2849334
[6,] 0.2963158 0.2849334

```

In any case, the third row shows us that K=10 did slightly better with a CV R-squared of 58.43% vs 56.77% for K=5.

Let's try K=10 vs K=15. Note, I will not be doing more than one repetition of LOOCV, because as discussed above, it's pointless - the results are the same every iteration.

```
[1] 0.5843193 0.5714809
```

We see that K=15 did worse, with a CV R-squared of 57.14%% Let's try K=10 and K=12

```
[1] 0.5843193 0.5768682
```

K=12 doesn't do better. Let's compare K=8 and K=10

```
[1] 0.5868482 0.5843193
```

K=8 does slightly better! Let's compare it with K=9

```
[1] 0.5868482 0.5854198
```

Again, K=8 is better. Let's compare with K=7

```
[1] 0.5848414 0.5868482
```

I conclude that K=8 is the best value for this model on this data. I already discussed the cons of using LOOCV for KNN - we can't use multiple replicates. The pro is that we are using as much of the training data as possible, so we are as close as possible to the way the model would be performing on completely new data.

b) What is the CV estimate for the prediction error Stand. Dev.?

I need to run LOOCV with K=8 (the best model).

I now have y & yhat for the K=8 model. The sd() of these two will be the prediction error standard deviation

```
[1] 0.5420608
```

c)

Now I fit the model on all the data, using $K=8$. These are the indices of the 8 nearest neighbours I need to standardize the values for the new observation as well.

```
New names:
* `` -> ...1
```

```
[1] 278 717 778 517 37 659 114 456
```

Now I compute the average \log_{10} _cost for these 8 neighbours (at the 8 indices in the data)

```
[1] 3.379489
```

These means that the cost for this individual would be $10^{3.482699}$, which is

```
[1] 2396.012
```

Problem 2 - GAM

I import the data, standardize the predictors and normalize the response. After talking with Suraj in Office Hours, I understand it's not necessary for me to normalize the response, but I will keep it for consistency across the HW.

```
New names:
* `` -> ...1
```

	age	gend	intvtn	drugs	ervis	comp
1	0.63394297	-0.5437618	-0.48382784	0.5200762	0.2179635	-0.2302054
2	0.04171172	-0.5437618	-0.48382784	-0.4198780	0.9762649	-0.2302054
3	0.48588516	-0.5437618	2.19729972	-0.4198780	-0.5403379	-0.2302054
4	0.18976953	1.8367066	0.76736502	-0.4198780	1.3554156	-0.2302054
5	-0.55051953	-0.5437618	0.05239767	1.4600303	1.3554156	-0.2302054
6	1.07811641	-0.5437618	-0.66256968	-0.4198780	-0.1611872	-0.2302054
	comorb	dur	log10_cost			
1	-0.1288017	1.1244965	0.4771970			
2	-0.6329195	-0.3641411	0.5302932			
3	0.2072769	1.5628176	0.8406152			
4	-0.2968410	1.3891432	0.5185938			
5	-0.6329195	-1.2077024	0.9048928			
6	0.0392376	1.0914157	0.5626325			

Fitting a GAM. Note that the gend, drugs, and comp are included as direct linear variables, because they don't have enough unique values for a smoother function. I tried turning them into factors and the results (in terms of variable significance) were the same.

```
Family: gaussian
Link function: identity
```

Formula:

```
log10_cost ~ s(age) + gend + s(intvn) + drugs + s(ervis) + comp +
  s(comorb) + s(dur)
```

Parametric coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.578567	0.003508	164.919	< 2e-16 ***
gend	-0.006302	0.003568	-1.766	0.0777 .
drugs	-0.006151	0.004393	-1.400	0.1619
comp	0.015215	0.003662	4.154	3.63e-05 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:

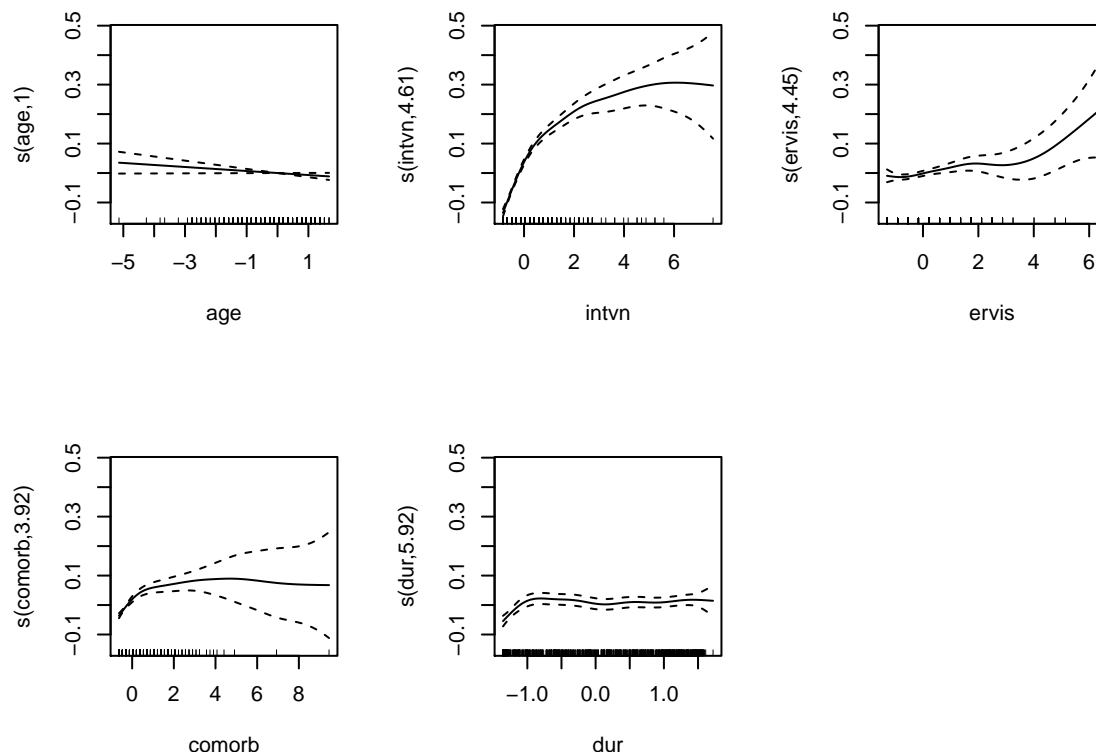
	edf	Ref.df	F	p-value
s(age)	1.000	1.000	3.617	0.05757 .
s(intvn)	4.610	5.563	136.168	< 2e-16 ***
s(ervis)	4.450	5.435	3.101	0.00606 **
s(comorb)	3.924	4.809	17.023	2.72e-15 ***
s(dur)	5.917	7.043	5.514	3.11e-06 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) = 0.685 Deviance explained = 69.4%

GCV = 0.010002 Scale est. = 0.0096983 n = 788

Now I will make some plots to interpret the results. Note that I will be comparing only the variables which are fitted within smoother functions. I won't compare with the ones included directly.



Looking at the output, I can make the same conclusions as previous reviews of this data have suggested. - Intvn has the highest impact b/c as we change the value of intvn, the value of the smoother function changes the

most. - `ervis` is also influential, while the other 3 not so much. It's important to note that GAMs don't capture interactions so our interpretation of the above graphs might not be 'air-tight'. Also, even though duration doesn't change the value of the smoothing function very much, it does however have the highest significant coefficient from the summary output above (value is 5.917). So smaller changes in the smoother function for duration will still have impact on y . We can compare the coefficients, because, we are working with standardized data.

b)

The model above had the following smoothing parameters (estimated by the function itself):

```
s(age)      s(intvn)      s(ervis)      s(comorb)      s(dur)
5.916940e+06 9.350833e-02 1.716813e-01 6.336668e-02 1.076431e-01
```

I will use the above model to make predictions in LOOCV, just like in problem 1 in order to get predictions for each out-of-bag sample. More specifically, when fitting the GAM model to the training data, I will use `sp = myGAM$sp`. The goal is to estimate the prediction error standard deviation.

I now have y & \hat{y} for the myGAM model. The `sd()` of these two will be the prediction error standard deviation

```
[1] 0.0999712
```

It is important to note that this is for the normalized response variable. If I need the non-normalized standard deviation, I could just skip normalizing the response variable or 'denormalize' y and \hat{y} before placing them in the `sd()` function.



What are the pros & cons of using n -fold CV vs 10-fold cv for GAMs? A general drawback of using n -fold CV is that it can be initialized only once. On the other hand, k -fold CV could give us a more robust estimate for model accuracy w.r.t. new data, which is less influenced by the randomness of the fold selection.

Another observation is related to the bias-variance tradeoff. When we use n -fold, we are fitting our model to more of the data, therefore we have lower bias than with k -fold. On the other hand, we will have more variance when we see new data due to overfitting.

In terms of specific drawbacks for GAM models, I think that using a k -fold model would give less data to the training data. If specific attributes don't have many unique values this is a problem for GAM and with k -fold that exacerbates the problem.

c) - Making a prediction for a new observation

Note that I standardize the input variables and make a prediction. That prediction lies in the range $[0,1]$, because I fit my GAM model with a normalized response variable. Therefore, to get a prediction in the original scale of the data I have to 'denormalize' the prediction. I do the above in some code below. The result end up being:

```
New names:
* `` -> ...1
New names:
* `` -> ...1

New names:
* `` -> ...1
```

```
1
3.556478
```

Therefore, our prediction for the cost is $10^{3.556478}$. It's not very close to our KNN prediction, but it's important to note that our predictions are very sensitive, because they are in log10 scale i.e. even small changes in log10_cost lead to big changes in actual \$ cost.

```
[1] 3601.455
```

Problem 3

Loading the relevant data

```
New names:
* `` -> ...1
```

	intvn	drugs	ervis	dur	log10_cost
1	-0.48382784	0.5200762	0.2179635	1.1244965	2.253096
2	-0.48382784	-0.4198780	0.9762649	-0.3641411	2.503791
3	2.19729972	-0.4198780	-0.5403379	1.5628176	3.968982
4	0.76736502	-0.4198780	1.3554156	1.3891432	2.448552
5	0.05239767	1.4600303	1.3554156	-1.2077024	4.272471
6	-0.66256968	-0.4198780	-0.1611872	1.0914157	2.656482

a) - Using 10-fold CV to find the optimal span & degree

Note that the problem doesn't specify I have to use n-fold CV. I will define a function called loessCV, which does 10-fold CV with 20 initializations of the folds and allows me to compare 3 models at a time. The output contains the average MSE and the R-squared from CV. I will run it within suppressWarnings() to ignore the many warning messages that appear in the output.

First I use degree = 1 and compare span = .1 , .2 , .3

```
[[1]]
[1] 0.2126047 0.2297136 0.2235166

[[2]]
[1] 0.000000e+00 0.000000e+00 1.000742e-16

[[3]]
[1] 0.6898374 0.6648777 0.6739184
```

It appears that span = .1 does best. Let's try some more values around .1: .08, .1, .12

```
[[1]]
[1] 0.2038042 0.2126047 0.2050483

[[2]]
[1] 0.000000e+00 0.000000e+00 6.206335e-17

[[3]]
[1] 0.7026762 0.6898374 0.7008611
```

span = .08 does better. Let's compare it with .04 and .06

```
[[1]]
[1] 0.2073225 0.1998958 0.1913870

[[2]]
[1] 0.000000e+00 0.000000e+00 3.925231e-17

[[3]]
[1] 0.6975434 0.7083780 0.7207913
```

.06 does slightly better. Let's look at values closer to it: .05,.06,.07

```
[[1]]
[1] 0.1977932 0.1998958 0.1864936

[[2]]
[1] 0.000000e+00 0.000000e+00 2.775558e-17

[[3]]
[1] 0.7114454 0.7083780 0.7279301
```

.05 does best. I will pick it as the best span value. Now I will check degree = 0, 1, 2

```
[[1]]
[1] 0.2480750 0.1977932 0.1129725

[[2]]
[1] 0 0 0

[[3]]
[1] 0.6380908 0.7114454 0.8351877
```

Degree = 2 appears to perform the best.

b) - Using C_p to find the best span

First I will find an estimate for RMSE (the loess\$s attribute is the RMSE)

```
[1] 0.01 NaN
[1] 0.02 NaN
[1] 0.03 54561.21
[1] 0.040000 1.076855
[1] 0.05000 1.48238
[1] 0.0600000 0.6766925
[1] 0.0700000 0.5638402
[1] 0.0800000 0.5327028
[1] 0.0900000 0.5530266
[1] 0.1000000 0.5456473
[1] 0.110000 0.536925
[1] 0.1200000 0.5835478
```

```

[1] 0.1300000 0.5856549
[1] 0.1400000 0.5921578
[1] 0.1500000 0.5816489
[1] 0.1600000 0.573483
[1] 0.1700000 0.5693933
[1] 0.1800000 0.5629726
[1] 0.1900000 0.5592792
[1] 0.2000000 0.5565185

```

Looking at the estimates, the best RMSE estimate is .533 (at $\lambda = .08$)

Now I will use that estimate to find the best span via C_p .

```

[1] 0.01 NaN
[1] 0.02 NaN
[1] 3.000000e-02 1.832835e+09
[1] 0.0400000 0.9207829
[1] 0.0500000 1.626695
[1] 0.0600000 0.4621666
[1] 0.0700000 0.3610904
[1] 0.0800000 0.3324228
[1] 0.0900000 0.3445151
[1] 0.1000000 0.3359909
[1] 0.1100000 0.3265252
[1] 0.1200000 0.3655811
[1] 0.1300000 0.366092
[1] 0.1400000 0.369645
[1] 0.1500000 0.3584987
[1] 0.1600000 0.3493459
[1] 0.1700000 0.3446184
[1] 0.1800000 0.337798
[1] 0.1900000 0.3336836
[1] 0.2000000 0.3303191

```

Looking at the output, it appears that $\text{span} = .08$ is the best choice (somewhere around the first local minimum at $C_p = 0.3324228$) This value is close to the one suggested by CV.

Now, I will repeat the two steps to find the best degree.

```

[1] 0.0000000 0.5044718
[1] 1.0000000 0.483318
[1] 2.0000000 0.4680941

```

The best RMSE estimate is at $\text{deg} = 2$ and is $\text{RMSE} = .468$ Now I use that to estimate best degree via C_p

```

[1] 0.0000000 0.263512
[1] 1.0000000 0.2570692
[1] 2.0000000 0.2865354

```

Unlike with CV, the best degree choice is $\text{deg} = 1$ based on C_p .

c) Prediction Standard Error

Recall that the optimal parameters for loess found via CV are $sp = .05$ and $degree = 2$.

I will now run `nfold-CV` to get an estimate for the prediction error standard deviation for that best model

And take standard deviation of $y - \hat{y}$

```
[1] 0.0999712
```

This standard deviation seems a bit high. Looking at the errors, # 14 appears unusual. At this stage, my only suspicion might be that this point is somewhat of an outlier i.e. somewhere close to the edges of the range for the predictor variables where the smoother degree 2 approximation doesn't work very well.

```
      [,1]
[1,] -0.063715934
[2,]  0.006640631
[3,]  0.005735067
[4,] -0.221205627
[5,]  0.321542226
[6,]  0.055852897
[7,]  0.043605444
[8,]  0.192927275
[9,]  0.061471332
[10,] -0.015326269
[11,]  0.129001933
[12,]  0.091865947
[13,]  0.093157776
[14,] -0.121222303
[15,]  0.114312589
```

I look at the 14th observation in the data, along with a few of the surrounding one. Some of the predictor values for the 14th observation are quite below their respective averages, but so is the cost. In any case, this patient 'stands out' from the rest.

	intvn	drugs	ervis	dur	log10_cost
10	-0.30508600	-0.4198780	-0.5403379	-0.6535984	2.353339
11	0.05239767	-0.4198780	-1.2986393	-0.9017046	3.336300
12	-0.84131152	-0.4198780	1.3554156	-1.1415407	2.778079
13	0.23113951	1.4600303	-0.5403379	-0.6453282	3.347252
14	-0.84131152	0.5200762	-0.9194886	-1.3565661	1.049218
15	-0.48382784	-0.4198780	-1.2986393	-1.3482959	2.838660

The C_p estimate for the prediction error standard deviation is `sqrt(Cp)`:

```
[1] 0.507937
```

This is much lower than the one derived from CV.

d) Making a prediction for a new observation

I will fit the model with $\text{degree}=1$ and $\text{span} = .05$ (based on the CV). When making a prediction, I will ignore the attributes of the new patient, which were not included in the model i.e. I will include only `intvn`, `drugs`, `dur`, and `ervis`.

I will also standardize the input variables with the mean and sd from the dataset.

```
New names:
* `` -> ...1
New names:
* `` -> ...1

      1
3.413035
```

Our answer is $10^{3.413035}$, which is pretty close to the one we got from KNN. The issue is that, since we are in log10 scale, even small deviations in our prediction lead to a large change in predicted cost.

```
[1] 2588.422
```

This answer is quite close to our KNN prediction.

Problem 4

Loading the relevant data

```
New names:
* `` -> ...1

      age      gend      intvn      drugs      ervis      comp
1  0.63394297 -0.5437618 -0.48382784  0.5200762  0.2179635 -0.2302054
2  0.04171172 -0.5437618 -0.48382784 -0.4198780  0.9762649 -0.2302054
3  0.48588516 -0.5437618  2.19729972 -0.4198780 -0.5403379 -0.2302054
4  0.18976953  1.8367066  0.76736502 -0.4198780  1.3554156 -0.2302054
5 -0.55051953 -0.5437618  0.05239767  1.4600303  1.3554156 -0.2302054
6  1.07811641 -0.5437618 -0.66256968 -0.4198780 -0.1611872 -0.2302054
      comorb      dur log10_cost
1 -0.1288017  1.1244965  0.4771970
2 -0.6329195 -0.3641411  0.5302932
3  0.2072769  1.5628176  0.8406152
4 -0.2968410  1.3891432  0.5185938
5 -0.6329195 -1.2077024  0.9048928
6  0.0392376  1.0914157  0.5626325
```

a) Using CV to find the best value of nterms

Similar to `loessCV`, I define a function called `pprCV`, which takes two numbers: `n1` & `n2` and does 20 initializations of 10-fold CV for PPR with `nterms = n1` and `n2` respectively. Returns the AvgMSE, AvgMSE SD, and R-squared for the CV. I start by comparing `n=5` and `n=10`. I wrap the function call in `suppressWarnings()`.

```
[[1]]  
[1] 0.01051777 0.01139565
```

```
[[2]]  
[1] 0.0002536192 0.0004020954
```

```
[[3]]  
[1] 0.6579385 0.6293879
```

I will now do 4 pairs: 4 & 5; 3 & 4, 2 & 3, 1 & 2

```
[[1]]  
[1] 0.01030671 0.01050353
```

```
[[2]]  
[1] 0.0002398042 0.0002136183
```

```
[[3]]  
[1] 0.6648026 0.6584017
```

```
[[1]]  
[1] 0.01005169 0.01029599
```

```
[[2]]  
[1] 0.0002482615 0.0002351846
```

```
[[3]]  
[1] 0.6730967 0.6651514
```

```
[[1]]  
[1] 0.009873263 0.009990596
```

```
[[2]]  
[1] 0.0000987457 0.0001826575
```

```
[[3]]  
[1] 0.6788993 0.6750834
```

```
[[1]]  
[1] 0.009810002 0.009911912
```

```
[[2]]  
[1] 4.589782e-05 1.635169e-04
```

```
[[3]]  
[1] 0.6809567 0.6776424
```

I would interpret $n_{\text{terms}}=1$ as a single non-parametric model which is a function of a linear combination of all the variables.

b)

```
Call:
ppr(formula = log10_cost ~ ., data = heart, nterms = 1)

Goodness of fit:
 1 terms
7.425113

Projection direction vectors ('alpha'):
      age      gend      intvn      drugs      ervis      comp
-0.02759078 -0.02686806  0.93275393 -0.07100287  0.10038411  0.15520531
      comorb      dur
 0.28667863  0.08423339

Coefficients of ridge terms ('beta'):
 term 1
0.1458987
```



Now I will run CV to get an estimate of the predicted error standard deviation for the bestPPR mode with `nterms = 1`

And here I calculate `sd(y-yhat)`

```
[1] 0.09878118
```

The standard deviation turns out to be pretty close to the one for loess. Again, it is important to note this is SD for the normalized `log10_cost`.

c) Making a prediction for a new observation

I convert the new patient's predictors to standardized format and use bestPPR to make a prediction.

```
New names:
* `` -> ...1
New names:
* `` -> ...1
```

The value I get is for the normalized `log10_cost`:

```
      1
0.7435393
```

This means that the actual predicted value for `log10_cost` is:

```
New names:
* `` -> ...1

      1
3.510637
```

In actual dollar terms that is $10^{3.510637}$:

```
[1] 3240.686
```

PPR predicts a bit of a higher cost for our new observation.

Problem 5

Importing in relevant data

```
New names:
* `` -> ...1
```

	RI	Na	Mg	Al	Si	K
1	0.8708258	0.2842867	1.2517037	-0.6908222	-1.12444556	-0.67013422
2	-0.2487502	0.5904328	0.6346799	-0.1700615	0.10207972	-0.02615193
3	-0.7196308	0.1495824	0.6000157	0.1904651	0.43776033	-0.16414813
4	-0.2322859	-0.2422846	0.6970756	-0.3102663	-0.05284979	0.11184428
5	-0.3113148	-0.1688095	0.6485456	-0.4104126	0.55395746	0.08117845
6	-0.7920739	-0.7566101	0.6416128	0.3506992	0.41193874	0.21917466

	Ca	Ba	Fe	type01
1	-0.1454254	-0.3520514	-0.5850791	1
2	-0.7918771	-0.3520514	-0.5850791	1
3	-0.8270103	-0.3520514	-0.5850791	1
4	-0.5178378	-0.3520514	-0.5850791	1
5	-0.6232375	-0.3520514	-0.5850791	1
6	-0.6232375	-0.3520514	2.0832652	1

a) - Using CV to find the best K for this classification task

I will repurpose the CV Function from problem 1 to do 10 initializations of 10-fold CV and return the misclassification rate for each initialization for both models as well as the average misclassification rate across all initializations.

```
[[1]]
      [,1]      [,2]
[1,] 0.5046729 0.5233645
[2,] 0.5046729 0.5233645
[3,] 0.5046729 0.5233645
[4,] 0.5000000 0.5233645
[5,] 0.5046729 0.5233645
[6,] 0.5093458 0.5280374
[7,] 0.5046729 0.5233645
[8,] 0.5093458 0.5233645
[9,] 0.5046729 0.5186916
[10,] 0.5046729 0.5280374

[[2]]
[1] 0.5051402 0.5238318
```

It appears that K=5 gives a lower misclass rate so it's doing better than K=10

5 & 8? -> 5 does better

```
[[1]]
      [,1]      [,2]
[1,] 0.5046729 0.5093458
[2,] 0.5093458 0.5280374
```

```

[3,] 0.5046729 0.5233645
[4,] 0.5093458 0.5233645
[5,] 0.5046729 0.5186916
[6,] 0.5046729 0.5140187
[7,] 0.5046729 0.5280374
[8,] 0.5046729 0.5186916
[9,] 0.5093458 0.5186916
[10,] 0.5000000 0.5140187

```

```

[[2]]
[1] 0.5056075 0.5196262

```

5 & 6? -> 5

```

[[1]]
      [,1]      [,2]
[1,] 0.5046729 0.5046729
[2,] 0.5046729 0.5046729
[3,] 0.5046729 0.5046729
[4,] 0.5000000 0.5093458
[5,] 0.5093458 0.5093458
[6,] 0.5046729 0.5093458
[7,] 0.5093458 0.5093458
[8,] 0.5093458 0.5000000
[9,] 0.5000000 0.5046729
[10,] 0.5046729 0.5093458

```

```

[[2]]
[1] 0.5051402 0.5065421

```

4 & 5? -> 4

```

[[1]]
      [,1]      [,2]
[1,] 0.5046729 0.5140187
[2,] 0.5046729 0.5046729
[3,] 0.5046729 0.5046729
[4,] 0.5000000 0.5000000
[5,] 0.5046729 0.5046729
[6,] 0.5000000 0.5093458
[7,] 0.4953271 0.5046729
[8,] 0.4953271 0.5000000
[9,] 0.5000000 0.5046729
[10,] 0.5046729 0.5093458

```

```

[[2]]
[1] 0.5014019 0.5056075

```

3 & 4? -> 4 does better!

```

[[1]]
      [,1]      [,2]

```

```
[1,] 0.5093458 0.5046729
[2,] 0.5093458 0.5000000
[3,] 0.5046729 0.5000000
[4,] 0.5046729 0.5000000
[5,] 0.5046729 0.5046729
[6,] 0.5000000 0.5046729
[7,] 0.5000000 0.4953271
[8,] 0.5000000 0.5000000
[9,] 0.4953271 0.5000000
[10,] 0.5046729 0.4953271
```

```
[[2]]
[1] 0.5032710 0.5004673
```

I conclude that K=4 is the best model for this dataset.

b) - Using CV to find the best GAM for this classification task

GAM doesn't have any tuning parameters (that I have to tune; the function tunes span for each smoother)

```
Family: binomial
Link function: logit
```

```
Formula:
type01 ~ s(RI) + s(Na) + s(Mg) + s(Al) + s(Si) + s(K) + s(Ca) +
s(Ba) + s(Fe)
```

```
Parametric coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)    4.215      12.034    0.35   0.726
```

```
Approximate significance of smooth terms:
```

	edf	Ref.df	Chi.sq	p-value
s(RI)	1.073	1.139	2.104	0.1450
s(Na)	4.889	5.831	5.715	0.3746
s(Mg)	1.000	1.000	0.149	0.6993
s(Al)	4.855	5.641	8.811	0.1801
s(Si)	1.000	1.000	0.003	0.9589
s(K)	2.693	3.307	1.342	0.7650
s(Ca)	7.221	7.476	14.307	0.0457 *
s(Ba)	1.400	1.649	0.395	0.6601
s(Fe)	2.391	2.907	6.406	0.0833 .

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
R-sq.(adj) = 0.705   Deviance explained = 70.4%
UBRE = -0.37236   Scale est. = 1           n = 214
```

s(RI)	s(Na)	s(Mg)	s(Al)	s(Si)
2.559798e-01	4.645712e-04	1.196347e+04	4.322553e-04	6.293691e+03
s(K)	s(Ca)	s(Ba)	s(Fe)	
1.642143e-04	1.027133e-06	9.754247e-03	2.942407e-02	

Here are the predictions from the model in binary format:

[illegible]

And finally I calculate the misclassification rate (on the entire dataset which is the training data)

```
[1] 0.07476636
```

I can also try to estimate this with CV (which is definitely a better idea than looking at training misclassification rate, which as expected is quite low) I will reduce the number of folds to 3 so that each fold has enough observations for the smoothers to work properly. Will initialize 3 times.

Warning in newton(lsp = lsp, X = G\$X, y = G\$y, Eb = G\$Eb, UrS = G\$UrS, L = G\$L, : Iteration limit reached without full convergence - check carefully

```
Warning in newton(lsp = lsp, X = G$X, y = G$y, Eb = G$Eb, UrS = G$UrS, L =
G$L, : Fitting terminated with step failure - check results carefully
```

```
Warning in newton(lsp = lsp, X = G$X, y = G$y, Eb = G$Eb, UrS = G$UrS, L =
G$L, : Iteration limit reached without full convergence - check carefully
```

```
Warning in newton(lsp = lsp, X = G$X, y = G$y, Eb = G$Eb, UrS = G$UrS, L =
G$L, : Fitting terminated with step failure - check results carefully
```

```
Warning in newton(lsp = lsp, X = G$X, y = G$y, Eb = G$Eb, UrS = G$UrS, L =
G$L, : Iteration limit reached without full convergence - check carefully
```

```
Warning in newton(lsp = lsp, X = G$X, y = G$y, Eb = G$Eb, UrS = G$UrS, L =
G$L, : Fitting terminated with step failure - check results carefully
```

```

          [,1]
[1,] 0.2056075
[2,] 0.2149533
[3,] 0.2336449

```

```
[1] 0.2180685
```

The first 3 values are the average misclassification rate for each of the 3 initializations of 3-fold CV. The final is the average misclassification rate. We can see that the GAM does much poorer than the KNN, which had $\sim 50\%$ misclass. rate.

c) Comparing KNN, GAM, and NN



The Neural Network in HW 2 for the fgl data had a misclassification rate of 26.8% (in my solution while in the solution provided on canvas it is 27.7%). Apparently, our GAM model does a bit better with a 21.8% misclassification. The KNN model didn't do very well. It had around 50% misclassification rate (from when I was fitting using Cross-Validation.)

So, overall, the GAM appears to be the best model out of the three.

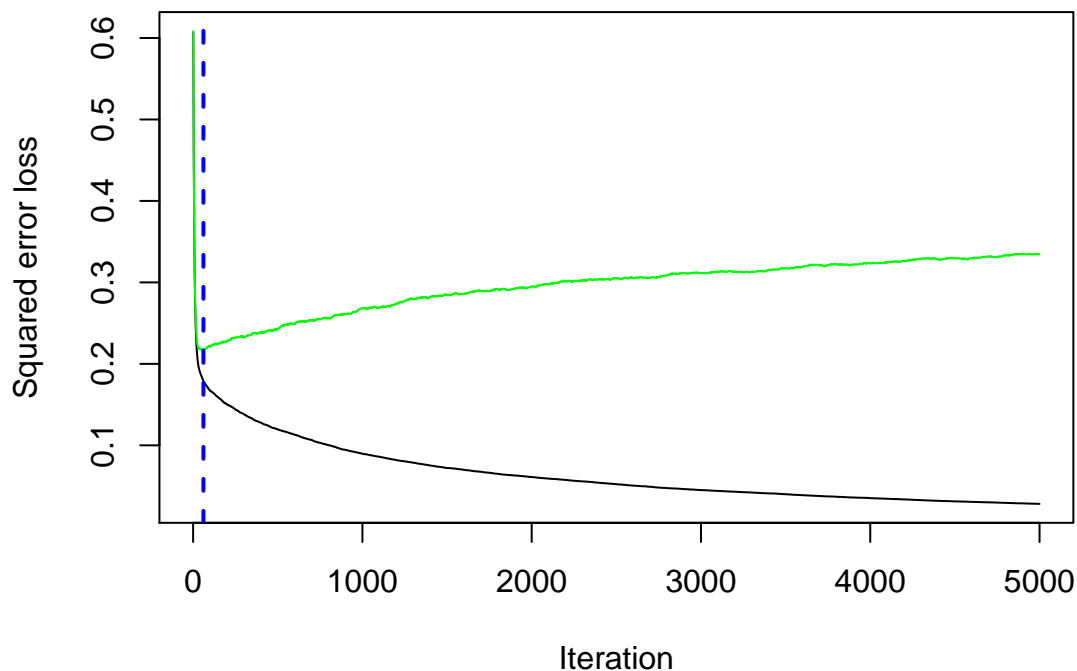
Problem 6 - Boosted Tree

Importing data

```
New names:
* `` -> ...1
```

	age	gend	intvn	drugs	ervis	comp	comorb	dur	log10_cost
1	63	0	2	1	4	0	3	300	2.253096
2	59	0	2	0	6	0	0	120	2.503791
3	62	0	17	0	2	0	5	353	3.968982
4	60	1	9	0	7	0	2	332	2.448552
5	55	0	5	2	7	0	0	18	4.272471
6	66	0	1	0	3	0	4	296	2.656482

Loaded gbm 2.1.5



Looking at the 60th to 70th CV error, I see that indeed the error at the 64th iteration is lowest

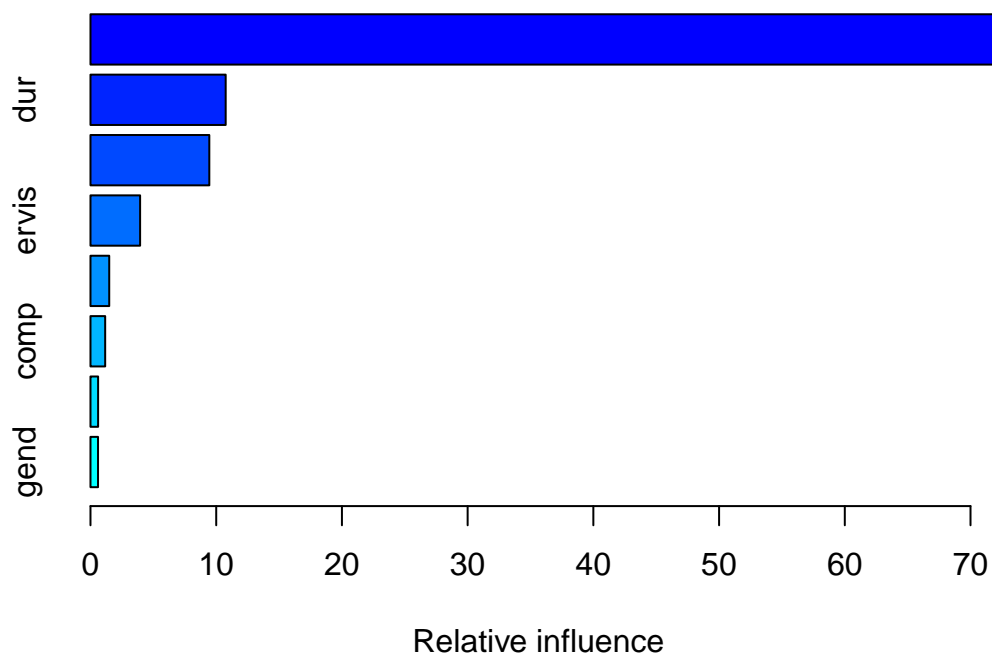
```
[1] 61
```

```
[1] 0.2173752 0.2169626 0.2172621 0.2174193 0.2174623 0.2183171 0.2184176
[8] 0.2180157 0.2176634 0.2179156 0.2180934
```

Below I calculate the R-squared for the Boosted Tree

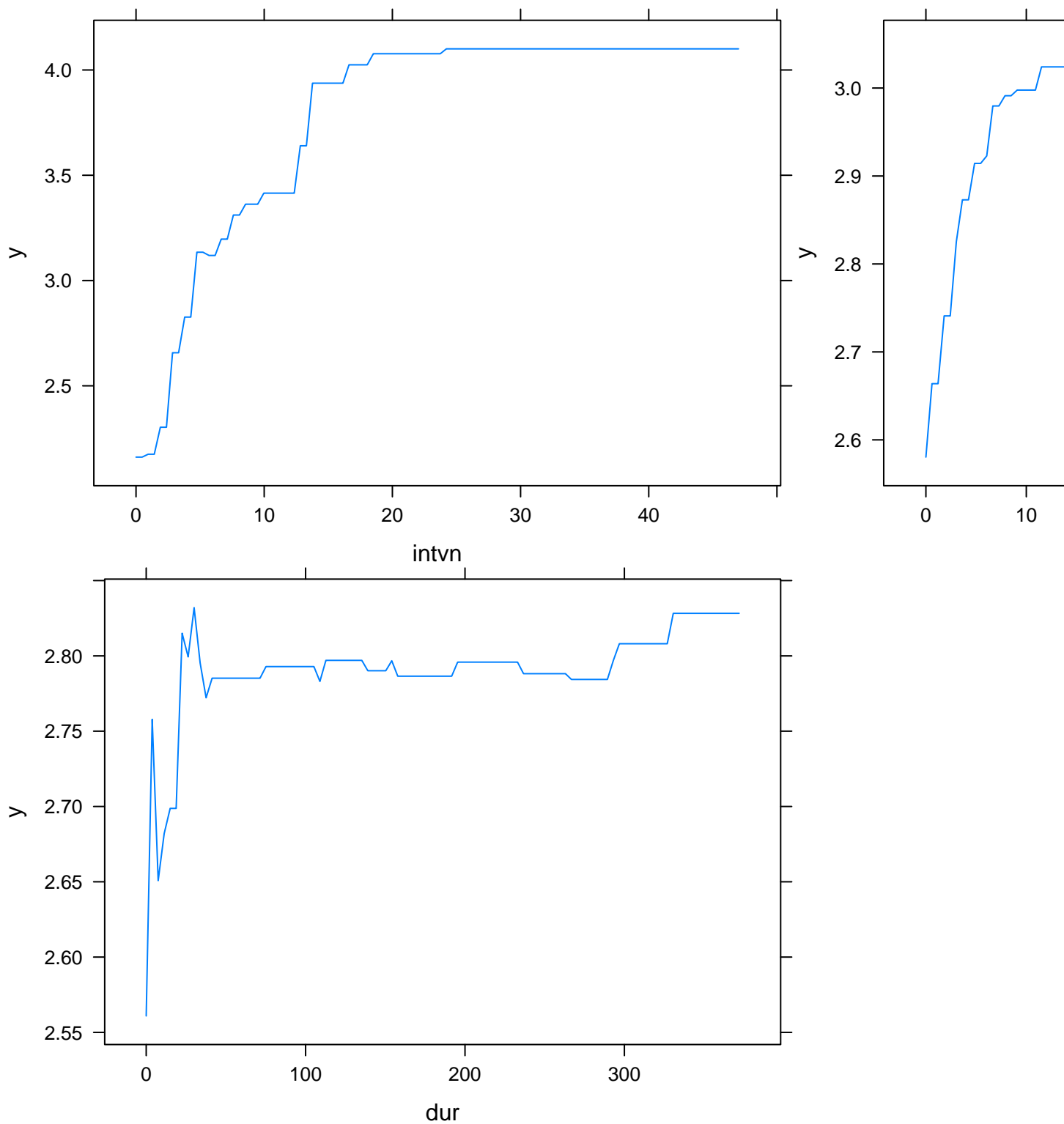
```
[1] 0.6834798
```

b) - Variable Importance

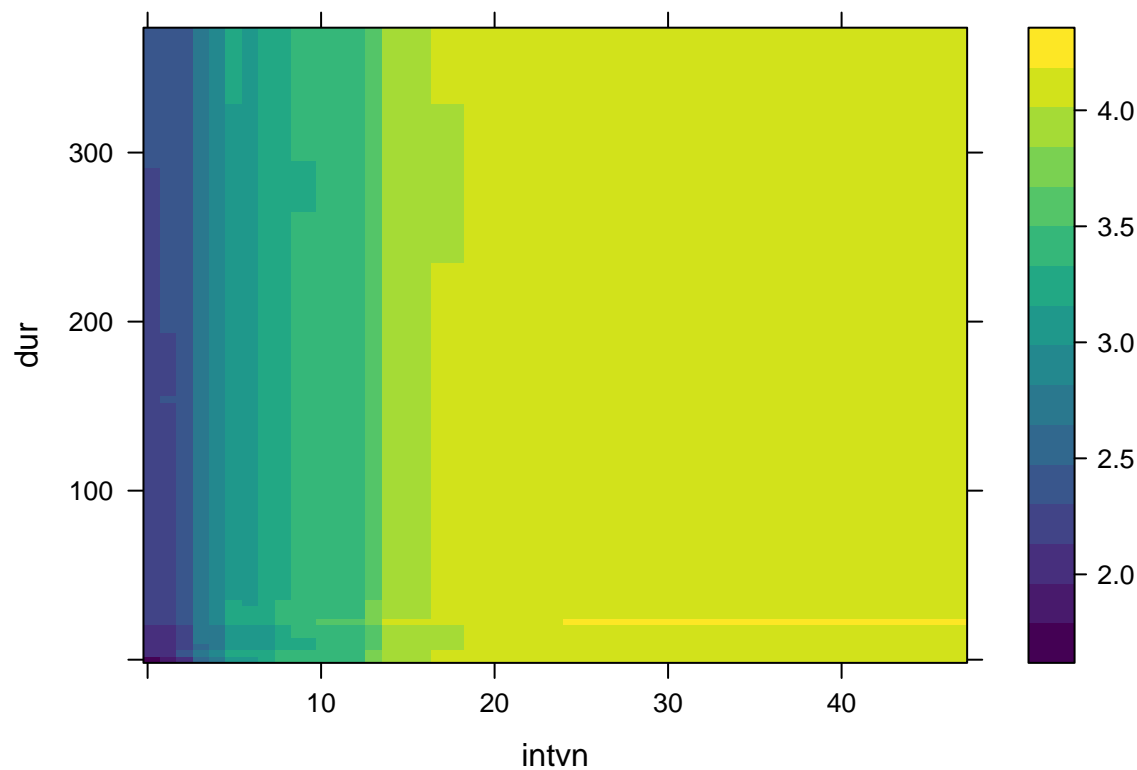


	var	rel.inf
intvn	intvn	71.9782828
dur	dur	10.7551534
comorb	comorb	9.4518373
eris	eris	3.9472409
age	age	1.4906663
comp	comp	1.1692351
drugs	drugs	0.6066330
gend	gend	0.6009511

The variable importance bar chart & accompanying table show us similar conclusions as before - intvn is the most important variable, followed by dur, comorb, and eris. This is supported based on the partial dependence plots below. In fact, the intvn plot looks much like the plot from the GAM model.



I'm just curious to see if there is an interaction b/w duration and intvn and there doesn't appear to be one based on the pairwise marginal plot below.



c) - Making a prediction

First I make a vector with the predictor variables

```
age gend intvn drugs ervis comp comorb dur
1  59   0   10    0     3    0     4  300
```

The log10_cost prediction is:

```
[1] 3.414972
```

Which translates to teh following in absolute dollar amount

```
[1] 2794.776
```

After speaking with Suraj during the lab, I will try to tune shrinkage & number of trees together via Grid Search in the caret package. I will try trees between 10 and 1000, with shrinkage between .001 and .1 in increments of .005. This means ~ 99,100 trees.

```
Loading required package: lattice
```

```
Loading required package: ggplot2
```

```
Attaching package: 'ggplot2'
```

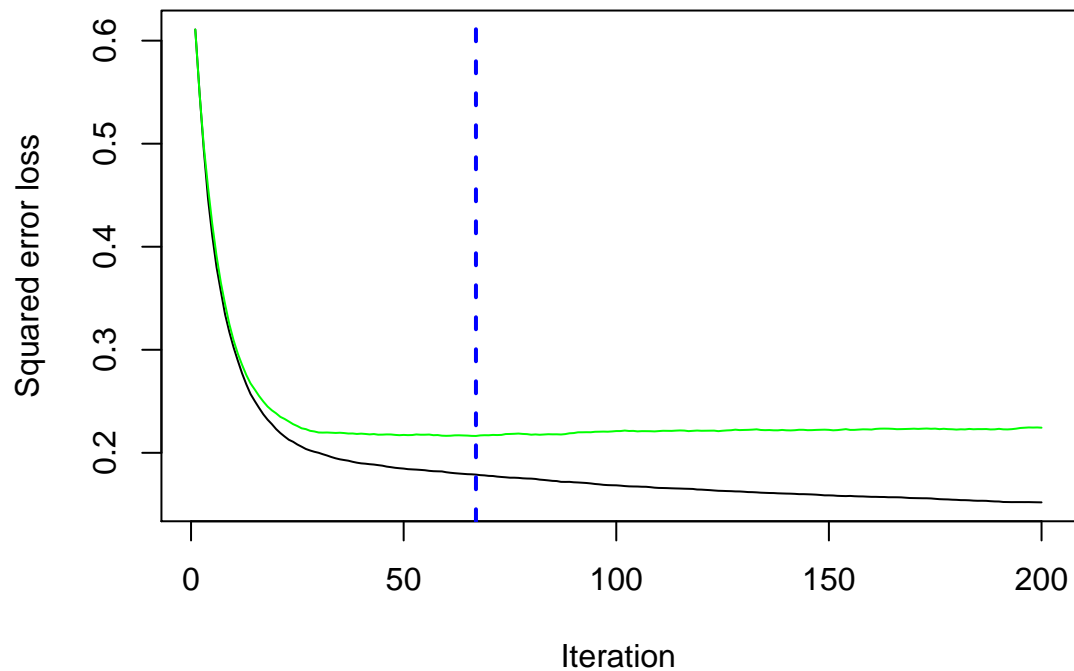
The following object is masked from 'package:yaImpute':

vars

	shrinkage	n.trees	n.minobsinnode	interaction.depth
1	0.001	10	10	3
2	0.006	10	10	3
3	0.011	10	10	3
4	0.016	10	10	3
5	0.021	10	10	3
6	0.026	10	10	3

n.trees	interaction.depth	shrinkage	n.minobsinnode
8068	149	3	0.041
			10

The best results turn out to be $n.trees = 59$, $shrinkage = .096$. The number of trees is very similar to the previous answer, but the shrinkage is much smaller. I'll now refit the model with these optimal parameters/



[1] 67

Below is the R-squared for this bestM = 43

[1] 0.684365

The R-Squared for $n.tree = 59$ is below and is very close to the one above. Difference is probably due to cross-validation variance in samples.

[1] 0.6839608

Problem 7 - Random Forests

a) - Fitting a tree with `mtry = 3` and `ntree = 500`

```
randomForest 4.6-14
```

```
Type rfNews() to see new features/changes/bug fixes.
```

```
Attaching package: 'randomForest'
```

```
The following object is masked from 'package:ggplot2':
```

```
margin
```

```
New names:
```

```
* `` -> ...1
```

	age	gend	intvtn	drugs	ervis	comp	comorb	dur	log10_cost
1	63	0	2	1	4	0	3	300	2.253096
2	59	0	2	0	6	0	0	120	2.503791
3	62	0	17	0	2	0	5	353	3.968982
4	60	1	9	0	7	0	2	332	2.448552
5	55	0	5	2	7	0	0	18	4.272471
6	66	0	1	0	3	0	4	296	2.656482

```
Call:
```

```
randomForest(formula = log10_cost ~ ., data = heart, mtry = 3, ntree = 500)
```

```
      Type of random forest: regression
```

```
      Number of trees: 500
```

```
No. of variables tried at each split: 3
```

```
      Mean of squared residuals: 0.2260458
```

```
      % Var explained: 66.98
```

I repeated the fitting multiple times and the value for % Var explained (R-squared) changes between 66.5 and 67.1 Below I've printed the R-squared for

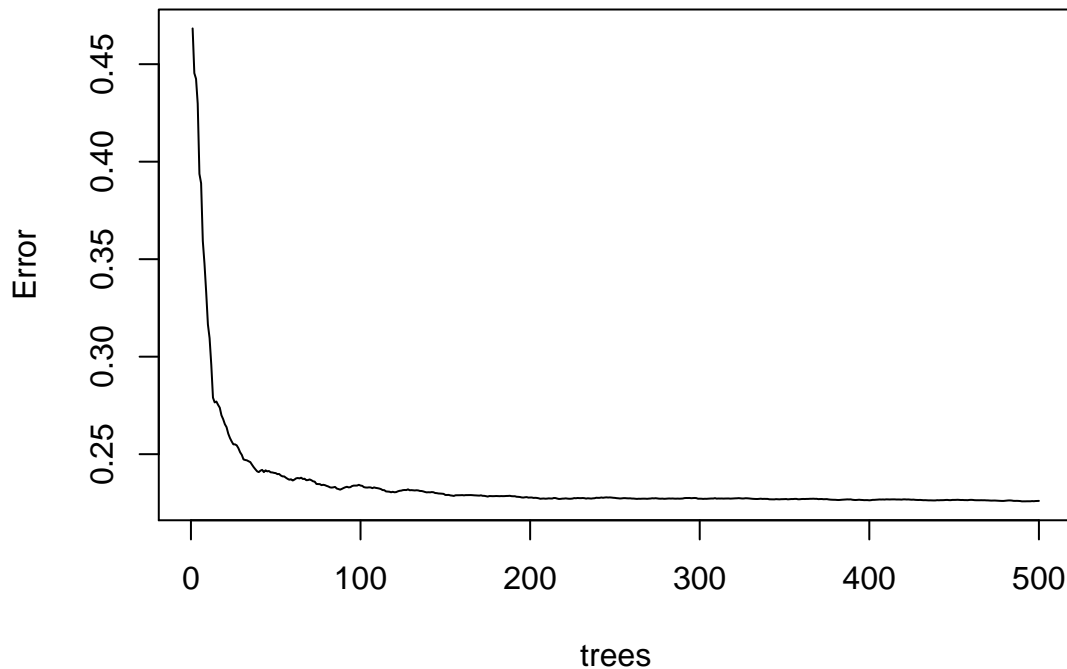
[1]	"call"	"type"	"predicted"
[4]	"mse"	"rsq"	"oob.times"
[7]	"importance"	"importanceSD"	"localImportance"
[10]	"proximity"	"ntree"	"mtry"
[13]	"forest"	"coefs"	"y"
[16]	"test"	"inbag"	"terms"

[1]	0.6697419	0.6696029	0.6694691	0.6694591	0.6695632	0.6697798	0.6698150
[8]	0.6697645	0.6698413	0.6699399	0.6701471	0.6701254	0.6701344	0.6700932
[15]	0.6701006	0.6700891	0.6699531	0.6700539	0.6699148	0.6698737	0.6698095

b) - Was `ntree = 500` enough?

I think 500 might be enough, particularly when I look at the below graph which shows the OOB error vs. the # of trees. There doesn't appear to be a significant decrease after 400 trees. But fitting more trees is not very computationally expensive, so typically I would fit more, just to check what happens.

myRF1



c) - Variable importance

	IncNodePurity
age	32.957637
gend	5.633971
intvn	248.126138
drugs	11.565457
ervis	43.600028
comp	9.221665
comorb	55.533483
dur	98.312531

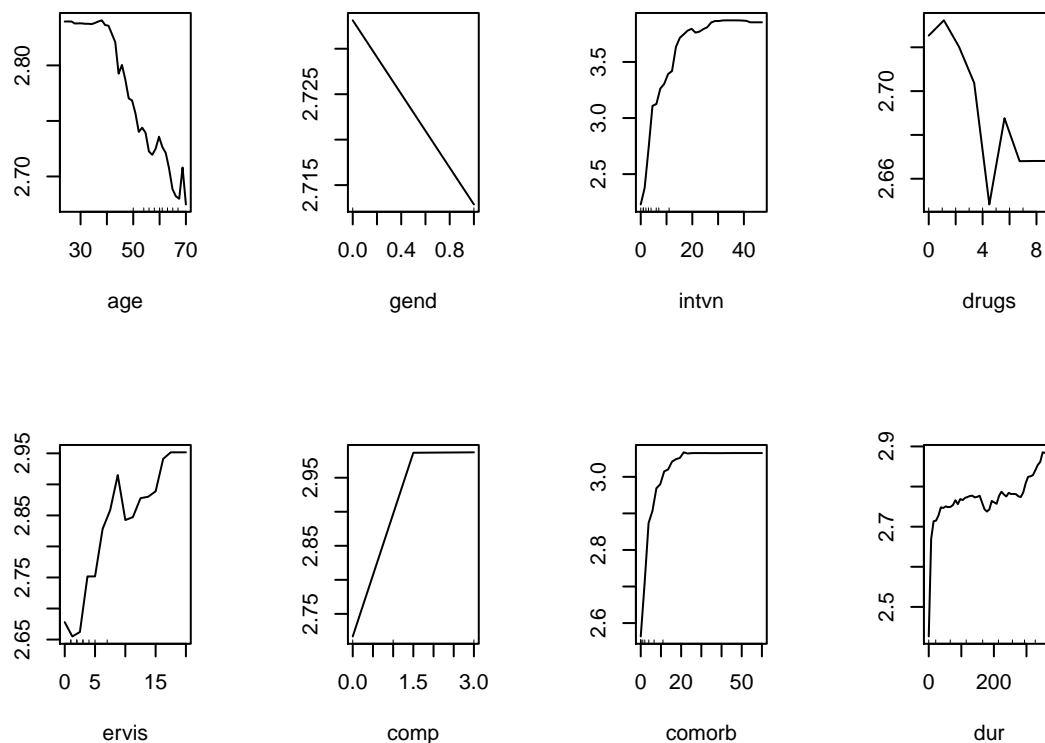
Once again, `intvn` appears to be the most important variable, in agreement with the result from the Boosted Tree. The follow-up variables are also the same: `dur`, `comorb`, and `ervis`.

d)

Looking at the partial dependence plots I take `intvn` as an example and observe: - the relationship is not entirely linear; there is definitely some 'curvature' or 'non-linearity' - the y axis changes the most for `intvn` vs the other variables, which once again suggests `intvn` is the most influential variable. This is in line with both what we saw from the variable importance metrics above and the same as the Boosted Tree. - Furthermore,

the PD plot for the boosted tree and for the RF are very similar - both in shape and the values on the y axis.

On another note: - drugs doesn't look linear - age, within the domain of x values (the tick marks in the PD plot) does not appear linear; same for ervis



e) Making a prediction for a new observation

```
age gend intvn drugs ervis comp comorb dur
1  59   0    10    0     3    0     4 300
```

I make a prediction

```
1
3.579941
```

Which in absolute \$ terms is:

```
[1] 3987.659
```

It appears much higher than the other models I've fit.

f)

First I do mtry=1


```

Call:
  randomForest(formula = log10_cost ~ ., data = heart, mtry = 1,      ntree = 500)
      Type of random forest: regression
      Number of trees: 500
No. of variables tried at each split: 1

      Mean of squared residuals: 0.2974659
      % Var explained: 56.55

Call:
  randomForest(formula = log10_cost ~ ., data = heart, mtry = 3,      ntree = 500)
      Type of random forest: regression
      Number of trees: 500
No. of variables tried at each split: 3

      Mean of squared residuals: 0.2268242
      % Var explained: 66.87

```

It appears that mtry=1 is much worse and mtry=2 is only a little worse than mtry=3

What is mtry? mtry are the number of variables we randomly pick from all of the variables we have when we are trying to determine what kind of split to make. The steps to making a RF are: - Draw a bootstrap sample - Take a subset of variables for that bootstrap sample (of length = mtry) - Split the root node by whatever variable and value produces the greatest reduction in SSE (or gini index/deviance in classification setting) - Continue repeating the previous two steps until we reach a condition such as minimum observations in a node or maximum tree size (depth)

The mtry parameter allows us to control how many variables will be considered at each step when trying to grow a tree i.e. splitting a leaf node.


g) - Model comparison

R-squared:

Random Forest: 67.18%

Boosted Tree: 69.08% (From the CV GridSearch with Caret)

PPR: 67.08% (From the final call of pprCV function, when I was tuning nterms and found nterms = 1 to be the best)

Loess (Local Regression with deg = 2 and sp = .05): 83.51% (From the final call of LoessCV, when I was tuning the degree parameter) 

GAM: I don't have an R-squared, because I didn't use CV for GAM (the function optimized the parameters)

KNN (K=8): 58.68% (from final call of LOOCV function)

Neural Network: 57.03%

Regression Tree: 63.5%

Overall, it looks like Loess did the best, followed closely by the Boosted Tree.

Appendix

This section is to be used for including your R code. The following lines of code will take care of it. Please make sure to comment your code appropriately - in particular, demarcating codes belonging to different questions. Among other things, it will be easier for you to debug your own code.

```

library(readxl) # To read in Excel
library(yaImpute) # To do K-Nearest Neighbours
library(mgcv)

set.seed(42)

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
}

## THIS FUNCTION USED REPEATEDLY IN PROBLEM 1 TO FIND BEST K for KNN on heart data
LOOCV_2KNN <- function(K1, K2){
  Nrep<-1 # number of replicates of CV
  K<-nrow(heart) # n-fold CV on each replicate i.e. LOOCV Leave One Out CV
  n.models = 2 #number of different models to fit
  n=nrow(heart)
  y<-heart$log10_cost
  yhat=matrix(0,n,n.models)
  MSE<-matrix(0,Nrep,n.models)
  for (j in 1:Nrep) {
    Ind<-CVInd(n,K)
    for (k in 1:K) {
      train<-as.matrix(heart[-Ind[[k]],1:8]) # Training data for Predictors
      test<-as.matrix(heart[Ind[[k]],1:8]) # Test data for Predictors
      ytrain<-heart[-Ind[[k]],9] # The response variable for the Training data
      out<-ann(train,test,K1,verbose=F) # Train the model
      ind<-as.matrix(out$knIndexDist[,1:K1]) # These are the nearest neighbours
      yhat[Ind[[k]],1]<-apply(ind,2,function(x) mean(ytrain[x])) # Look at the training data at the indi
      out<-ann(train,test,K2,verbose=F)
      ind<-as.matrix(out$knIndexDist[,1:K2])
      yhat[Ind[[k]],2]<-apply(ind,2,function(x) mean(ytrain[x]))
    } #end of k loop
    MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
  } #end of j loop
  MSEAve<- apply(MSE,2,mean); MSEAve #averaged mean square CV error
  MSEsd <- apply(MSE,2,sd); #SD of mean square CV error
  r2<-1-MSEAve/var(y); r2 #CV r^2
}

## THIS FUNCTION USED REPEATEDLY IN PROBLEM 3 TO FIND BEST SPAN & DEGREE
loessCV = function(span, degree){

```

```

Nrep<-2 # number of replicates of CV
K<-10 # n-fold CV on each replicate i.e. LOOCV Leave One Out CV
n.models = 3 #number of different models to fit
n=nrow(heart)
y<-heart$log10_cost
yhat=matrix(0,n,n.models)
MSE<-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {
    train<-heart[-Ind[[k]],] # Training data
    test<-heart[Ind[[k]],1:4] # Test data for Predictors

    # Model 1
    out1<-loess(log10_cost ~., data=heart,degree=degree[1], span=span[1]); # Train the model
    yhat[Ind[[k]],1] = predict(out1,test)
    # Model 2
    out2<-loess(log10_cost ~., data=heart,degree=degree[2], span=span[2]); # Train the model
    yhat[Ind[[k]],2] = predict(out2,test)
    # Model 3
    out3<-loess(log10_cost ~., data=heart,degree=degree[3], span=span[3],control=loess.control(surface="all"))
    yhat[Ind[[k]],3] = predict(out3,test)
  } #end of k loop
  MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
} #end of j loop
MSEAve<- apply(MSE,2,mean); #averaged mean square CV error
MSEsd <- apply(MSE,2,sd); #SD of mean square CV error
r2<-1-MSEAve/var(y); #CV r^2
lst = list(MSEAve, MSEsd, r2)
return(lst)
}

##### QUESTION 1 begins here #####
# loading data for question 1
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart <- as.data.frame(heart)
heart[,1:8] = scale(heart[,1:8]) # Standardizing the predictor variables!
head(heart)
Nrep<-20 # number of replicates of CV
K<-nrow(heart) # n-fold CV on each replicate i.e. LOOCV Leave One Out CV
n.models = 2 #number of different models to fit
n=nrow(heart)
y<-heart$log10_cost
yhat=matrix(0,n,n.models)
MSE<-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {
    train<-as.matrix(heart[-Ind[[k]],1:8]) # Training data for Predictors
    test<-as.matrix(heart[Ind[[k]],1:8]) # Test data for Predictors
    ytrain<-heart[-Ind[[k]],9] # The response variable for the Training data

```

```

K1=5;K2=10
out<-ann(train,test,K1,verbose=F) # Train the model
ind<-as.matrix(out$knIndexDist[,1:K1]) # These are the nearest neighbours
yhat[Ind[[k]],1]<-apply(ind,2,function(x) mean(ytrain[x])) # Look at the training data at the indi
out<-ann(train,test,K2,verbose=F)
ind<-as.matrix(out$knIndexDist[,1:K2])
yhat[Ind[[k]],2]<-apply(ind,2,function(x) mean(ytrain[x]))
} #end of k loop
MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
} #end of j loop
MSEAve<- apply(MSE,2,mean); MSEAve #averaged mean square CV error
MSEsd <- apply(MSE,2,sd); MSEsd #SD of mean square CV error
r2<-1-MSEAve/var(y); r2 #CV r^2
head(MSE)
LOOCV_2KNN(10,15)
LOOCV_2KNN(10,12)
LOOCV_2KNN(8,10)
LOOCV_2KNN(8,9)
LOOCV_2KNN(7,8)
K<-nrow(heart) # n-fold CV on each replicate i.e. LOOCV Leave One Out CV
n.models = 1 #number of different models to fit
n=nrow(heart)
y<-heart$log10_cost
yhat=matrix(0,n,n.models)
Ind<-CVInd(n,K)

for (k in 1:K) {
  train<-as.matrix(heart[-Ind[[k]],1:8]) # Training data for Predictors
  test<-as.matrix(heart[Ind[[k]],1:8]) # Test data for Predictors
  ytrain<-heart[-Ind[[k]],9] # The response variable for the Training data
  K1=5;
  out<-ann(train,test,K1,verbose=F) # Train the model
  ind<-as.matrix(out$knIndexDist[,1:K1]) # These are the nearest neighbours
  yhat[Ind[[k]],1]<-apply(ind,2,function(x) mean(ytrain[x])) # Look at the training data at the indi
} #end of k loop
sd(y-yhat)
predictors<-as.matrix(heart[,1:8]) # This is the standardized data

## NEED ORIGINAL HEART DATA TO STANDARDIZE THE PREDICTOR VALUES FOR THE NEW OBSERVATION
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart <- as.data.frame(heart)

## Let's try to standardize the input data
ageMean = mean(heart$age)
gendMean = mean(heart$gend)
intvnMean = mean(heart$intvn)
drugsMean = mean(heart$drugs)
ervisMean = mean(heart$ervis)
compMean = mean(heart$comp)
comorbMean = mean(heart$comorb)
durMean = mean(heart$dur)

```

```

ageSD = sd(heart$age)
gendSD = sd(heart$gend)
intvnSD = sd(heart$intvn)
drugsSD = sd(heart$drugs)
ervisSD = sd(heart$ervis)
compSD = sd(heart$comp)
comorbSD = sd(heart$comorb)
durSD = sd(heart$dur)

ageNew = (59-ageMean)/ageSD
gendNew = (0-gendMean)/gendSD
intvnNew = (10-intvnMean)/intvnSD
drugsNew = (0-drugsMean)/drugsSD
ervisNew = (3-ervisMean)/ervisSD
compNew = (0-compMean)/compSD
comorbNew = (4-comorbMean)/comorbSD
durNew = (300-durMean)/durSD

response = t(c(ageNew, gendNew,intvnNew,drugsNew,ervisNew,compNew,comorbNew,durNew))
# names(response) <- names(heart)[1:8]

# response<- t(c(59,0,10,0,3,0,4,300))
K = 8
bestKNN <- ann(predictors,response,K,verbose=F) # Train the model on the standardized data
indices = bestKNN$knIndexDist[,1:8] # These are the indices of the 8 nearest neighbours
indices
mean(heart$log10_cost[indices])
10^3.379489
##### QUESTION 2 begins here #####
# loading data for question 2
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart <- as.data.frame(heart)
heart[,c(1:8)] = scale(heart[,c(1:8)]) # Standardizing the predictor variables!
heart[9] = (heart[9] - min(heart[9]))/(max(heart[9])-min(heart[9])) # Apley appears to be normalizing t
head(heart)
myGAM<-gam(log10_cost~s(age) + gend + s(intvn) + drugs + s(ervis) + comp + s(comorb) + s(dur), data=hea
summary(myGAM)
par(mfrow=c(2,3))
plot(myGAM)
myGAM$sp
K<-nrow(heart) # n-fold CV on each replicate i.e. LOOCV Leave One Out CV
n.models = 1 #number of different models to fit
n=nrow(heart)
y<-heart$log10_cost
yhat=matrix(0,n,n.models)
Ind<-CVInd(n,K)
for (k in 1:K) {
  train<-heart[-Ind[[k]],] # Training data
  test<-heart[Ind[[k]],1:8] # Test data for Predictors
  out<-gam(log10_cost~s(age) + gend + s(intvn) + drugs + s(ervis) + comp + s(comorb) + s(dur), data=
  yhat[Ind[[k]],1] = predict(out,test)

```

```

}
sd(y-yhat)
# loading data for question 2
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart <- as.data.frame(heart)
# heart[,c(1:8)] = scale(heart[,c(1:8)]) # Standardizing the predictor variables!
heart[9] = (heart[9] - min(heart[9]))/(max(heart[9])-min(heart[9])) # Apley appears to be normalizing t
# head(heart)

## Let's try to standardize the input data
ageMean = mean(heart$age)
gendMean = mean(heart$gend)
intvnMean = mean(heart$intvn)
drugsMean = mean(heart$drugs)
ervisMean = mean(heart$ervis)
compMean = mean(heart$comp)
comorbMean = mean(heart$comorb)
durMean = mean(heart$dur)

ageSD = sd(heart$age)
gendSD = sd(heart$gend)
intvnSD = sd(heart$intvn)
drugsSD = sd(heart$drugs)
ervisSD = sd(heart$ervis)
compSD = sd(heart$comp)
comorbSD = sd(heart$comorb)
durSD = sd(heart$dur)

ageNew = (59-ageMean)/ageSD
gendNew = (0-gendMean)/gendSD
intvnNew =(10-intvnMean)/intvnSD
drugsNew = (0-drugsMean)/drugsSD
ervisNew = (3-ervisMean)/ervisSD
compNew = (0-compMean)/compSD
comorbNew = (4-comorbMean)/comorbSD
durNew = (300-durMean)/durSD

response = as.data.frame(t(c(ageNew, gendNew,intvnNew,drugsNew,ervisNew,compNew,comorbNew,durNew)))
names(response) <- names(heart)[1:8]

heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart <- as.data.frame(heart)
heart[,c(1:8)] = scale(heart[,c(1:8)]) # Standardizing the predictor variables!
heart[9] = (heart[9] - min(heart[9]))/(max(heart[9])-min(heart[9])) # Apley appears to be normalizing t
# head(heart)
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart <- as.data.frame(heart)

```

```

# Finally, we need to 'denormalize' the result i.e. the log10_cost response variable used to be normali
# The prediction is ~.75, which I now convert to the original scale of the log10_cost variable.
(predict(myGAM, response))*(max(heart[9])-min(heart[9]))+min(heart[9])
10^3.556478
##### QUESTION 3 begins here #####
# loading data for question 3
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart = heart[,c(3,4,5,8,9)] # Using only the relevant predictors: intvn, drugs, ervis, dur
heart[,1:4] = scale(heart[,1:4]) # Standardizing the predictor variables!
heart <- as.data.frame(heart)
head(heart)
sp1 = c(.1, .2, .3)
deg1 = c(1,1,1)
suppressWarnings(loessCV(sp1,deg1))
sp1 = c(.08, .1, .12)
deg1 = c(1,1,1)
suppressWarnings(loessCV(sp1,deg1))
sp1 = c(.04, .06, .08)
deg1 = c(1,1,1)
suppressWarnings(loessCV(sp1,deg1))
sp1 = c(.05, .06, .07)
deg1 = c(1,1,1)
suppressWarnings(loessCV(sp1,deg1))
sp1 = c(.05, .05, .05)
deg1 = c(0,1,2)
suppressWarnings(loessCV(sp1,deg1))
##first find sigma_hat for a low-bias model###
suppressWarnings(for (lambda in seq(.01,.2,.01)) {
  out<-loess(log10_cost ~., data=heart, degree=2, span=lambda)
  print(c(lambda,out$s))
})
sig_hat<-.533
##now find Cp for various lambda###
suppressWarnings(
  for (lambda in c(seq(.01,.2,.01))) {
    out<-loess(log10_cost ~., heart, degree=2, span=lambda)
    SSE<-sum((heart[,5]-out$fitted)^2)
    Cp <- (SSE+2*out$trace.hat*sig_hat^2)/nrow(heart)
    print(c(lambda,Cp))
  }
)
##first find sigma_hat for a low-bias model###
suppressWarnings(for (degree in c(0,1,2)) {
  out<-loess(log10_cost ~., data=heart, degree=degree, span=.08,control=loess.control(surface = "direct")
  print(c(degree,out$s))
})
sig_hat = .49
##now find Cp for various lambda###
suppressWarnings(
  for (deg in c(0,1,2)) {
    out<-loess(log10_cost ~., heart, degree=deg, span=.08,control=loess.control(surface = "direct"))

```

```

    SSE<-sum((heart[,5]-out$fitted)^2)
    Cp <- (SSE+2*out$trace.hat*sig_hat^2)/nrow(heart)
    print(c(deg,Cp))
  }
)
predErrorCV = function(){
  K<-nrow(heart)  # n-fold CV on each replicate i.e. LOOCV Leave One Out CV
  n.models = 1
  n=nrow(heart)
  y<-heart$log10_cost
  yhat=matrix(0,n,n.models)

  Ind<-CVInd(n,K)
  for (k in 1:K) {
    train<-heart[-Ind[[k]],] # Training data
    test<-heart[Ind[[k]],1:4] # Test data for Predictors

    # Model 1
    out1<-loess(log10_cost ~., data=train,degree=2, span=.05); # Train the model
    yhat[Ind[[k]],1] = predict(out1,test)

  } #end of k loop
}
suppressWarnings(
  predErrorCV()
)
sd(y-yhat)
head(y-yhat,15)
heart[10:15,]
sqrt(.258)
### Reloading the data so that I can get the Means and Standard Deviations
# In order to standardize the values for the new prediction
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart = heart[,c(3,4,5,8,9)] # Using only the relevant predictors: intvn, drugs, ervis, dur
# heart[,1:4] = scale(heart[,1:4]) # Standardizing the predictor variables!
heart <- as.data.frame(heart)

## Let's try to standardize the input data
intvnMean = mean(heart$intvn)
drugsMean = mean(heart$drugs)
ervisMean = mean(heart$ervis)
durMean = mean(heart$dur)

intvnSD = sd(heart$intvn)
drugsSD = sd(heart$drugs)
ervisSD = sd(heart$ervis)
durSD = sd(heart$dur)

intvnNew =(10-intvnMean)/intvnSD
drugsNew = (0-drugsMean)/drugsSD
ervisNew = (3-ervisMean)/ervisSD

```



```

durNew = (300-durMean)/durSD

response = as.data.frame(t(c(intvnNew,drugsNew,ervisNew,durNew)))
names(response) <- names(heart)[1:4]

# Finally, in order to fit the model I reimport the full dataset
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart = heart[,c(3,4,5,8,9)] # Using only the relevant predictors: intvn, drugs, ervis, dur
heart[,1:4] = scale(heart[,1:4]) # Standardizing the predictor variables!
heart <- as.data.frame(heart)
suppressWarnings(out<-loess(log10_cost ~., heart, degree=2, span=.05,control=loess.control(surface = "d
suppressWarnings(predict(out, response))
10^3.413035
##### QUESTION 4 begins here #####
# loading data for question 4
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart[,c(1:8)] = scale(heart[,c(1:8)]) # Standardizing the predictor variables!
heart[9] = (heart[9] - min(heart[9]))/(max(heart[9])-min(heart[9])) # Apley appears to be normalizing t
heart <- as.data.frame(heart)
head(heart)
pprCV = function(n1,n2){

  Nrep<-20 # number of replicates of CV
  K<-10
  n.models = 2 #number of different models to fit
  n=nrow(heart)
  y<-heart$log10_cost
  yhat=matrix(0,n,n.models)
  MSE<-matrix(0,Nrep,n.models)
  for (j in 1:Nrep) {
    Ind<-CVInd(n,K)
    for (k in 1:K) {
      train<-heart[-Ind[[k]],] # Training data
      test<-heart[Ind[[k]],1:8] # Test data for Predictors

      out1<- ppr(log10_cost ~., data=train, nterms=n1)# Train the model
      yhat[Ind[[k]],1] = predict(out1,test)

      out2<- ppr(log10_cost ~., data=train, nterms=n2)# Train the model
      yhat[Ind[[k]],2] = predict(out2,test)

    } #end of k loop
    MSE[j,]=apply(yhat,2,function(x) sum((y-x)^2))/n
    # print(MSE)
  } #end of j loop

  MSEAve<- apply(MSE,2,mean); #averaged mean square CV error
  MSEsd <- apply(MSE,2,sd); #SD of mean square CV error

```

```

    r2<-1-MSEave/var(y);  #CV  $r^2$ 
    return(list(MSEave,MSEsd,r2))
}

suppressWarnings(pprCV(5,10))

suppressWarnings(pprCV(4,5))
suppressWarnings(pprCV(3,4))
suppressWarnings(pprCV(2,3))
suppressWarnings(pprCV(1,2))
bestPPR = ppr(log10_cost ~., data=heart, nterms=1)
summary(bestPPR)
K<-nrow(heart)
n.models = 1
n=nrow(heart)
y<-heart$log10_cost
yhat=matrix(0,n,n.models)

Ind<-CVInd(n,K)
for (k in 1:K) {
  train<-heart[-Ind[[k]],] # Training data
  test<-heart[Ind[[k]],1:8] # Test data for Predictors

  out1<- ppr(log10_cost ~., data=train, nterms=1)# Train the model
  yhat[Ind[[k]],1] = predict(out1,test)

} #end of k loop

sd(y-yhat)
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart <- as.data.frame(heart)
# heart[,c(1:8)] = scale(heart[,c(1:8)]) # Standardizing the predictor variables!
heart[9] = (heart[9] - min(heart[9]))/(max(heart[9])-min(heart[9])) # Apley appears to be normalizing t
# head(heart)

## Let's try to standardize the input data
ageMean = mean(heart$age)
gendMean = mean(heart$gend)
intvnMean = mean(heart$intvn)
drugsMean = mean(heart$drugs)
ervisMean = mean(heart$ervis)
compMean = mean(heart$comp)
comorbMean = mean(heart$comorb)
durMean = mean(heart$dur)

ageSD = sd(heart$age)
gendSD = sd(heart$gend)
intvnSD = sd(heart$intvn)
drugsSD = sd(heart$drugs)
ervisSD = sd(heart$ervis)
compSD = sd(heart$comp)

```

```

comorbSD = sd(heart$comorb)
durSD = sd(heart$dur)

ageNew = (59-ageMean)/ageSD
gendNew = (0-gendMean)/gendSD
intvnNew = (10-intvnMean)/intvnSD
drugsNew = (0-drugsMean)/drugsSD
ervisNew = (3-ervisMean)/ervisSD
compNew = (0-compMean)/compSD
comorbNew = (4-comorbMean)/comorbSD
durNew = (300-durMean)/durSD

response = as.data.frame(t(c(ageNew, gendNew, intvnNew, drugsNew, ervisNew, compNew, comorbNew, durNew)))
names(response) <- names(heart)[1:8]

heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart[,c(1:8)] = scale(heart[,c(1:8)]) # Standardizing the predictor variables!
heart[9] = (heart[9] - min(heart[9]))/(max(heart[9])-min(heart[9])) # Apley appears to be normalizing t
heart <- as.data.frame(heart)
# head(heart)
predict(bestPPR, response)
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart <- as.data.frame(heart)

# Finally, we need to 'denormalize' the result i.e. the log10_cost response variable used to be normali
# The prediction is ~.75, which I now convert to the original scale of the log10_cost variable.
(predict(bestPPR, response))*(max(heart[9])-min(heart[9]))+min(heart[9])
10^3.510637
##### QUESTION 5 begins here #####
# loading data for question 5
fgl = read_excel('data.xls', sheet = 'FGL data')
fgl = fgl[, -1]
# Defining the binary response
z<- (fgl$type == "WinF") | (fgl$type == "WinNF")
y<-as.character(fgl$type)
y[z]<-"Win"; y[!z]<-"Other"
y[y == "Win"]<-1; y[y == "Other"]<-0;
fgl<-data.frame(fgl, "type01"=as.numeric(y)) #also add a binary numeric response column
fgl = fgl[, -10]
fgl[,c(1:9)] = scale(fgl[,c(1:9)]) # Standardizing the data
fgl = as.data.frame(fgl)
head(fgl)
set.seed(42)
### Will use this function repeatedly to find the best K
classificationCV_2KNN <- function(K1, K2){
  Nrep<-10 # number of replicates of CV
  K<-10
  n.models = 2 #number of different models to fit
  n=nrow(fgl)

```

```

y<-fgl$type01
yhat=matrix(0,n,n.models)
misclass<-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {
    train<-as.matrix(fgl[-Ind[[k]],1:9]) # Training data for Predictors
    test<-as.matrix(fgl[Ind[[k]],1:9]) # Test data for Predictors
    ytrain<-fgl[-Ind[[k]],9] # The response variable for the Training data

    # Model 1
    out<-ann(train,test,K1,verbose=F) # Train the model; i.e. find the nearest neighbours in the tra
    ind<-as.matrix(out$knnIndexDist[,1:K1]) # These are the nearest neighbours
    # Look at the training data at the indices of the closest neighbours
    # Then compute the mean and return 1 if it's greater than .5 (which means there are more window
    # Else return 0, which means Other type of glass (not Window) as the prediction for the k-th fol

    yhat[Ind[[k]],1]<-apply(ind,1,function(x) ifelse(mean(ytrain[x])>=.5,1,0))
    # Model 2
    out<-ann(train,test,K2,verbose=F)
    ind<-as.matrix(out$knnIndexDist[,1:K2])
    yhat[Ind[[k]],2]<-apply(ind,1,function(x) ifelse(mean(ytrain[x])>=.5,1,0))
  } #end of k loop
  misclass[j,]=apply(yhat,2,function(x) sum(y != x)/n)
} #end of j loop
misclassAve<- apply(misclass,2,mean) #averaged CV misclass rate
return(list(misclass, misclassAve))
}

classificationCV_2KNN(5,10)
classificationCV_2KNN(5,8)
classificationCV_2KNN(5,6)
classificationCV_2KNN(4,5)
classificationCV_2KNN(3,4)
# fgl
bestGAMfgl<-gam(type01~s(RI) + s(Na) + s(Mg) + s(Al) + s(Si) + s(K) + s(Ca) + s(Ba) + s(Fe), data=fgl,
summary(bestGAMfgl)
bestGAMfgl$sp
# Initialize empty vector of length 214
bestGAMfgl01predictions = c(rep(0,nrow(fgl)))
# "Binarizing" the predictions
# NOTE: I am using type = 'response' in the prediction call so I get probabilities and NOT log-odds
for (i in 1:length(predict(bestGAMfgl))){
  if (predict(bestGAMfgl,type = 'response')[i]>=.5){
    bestGAMfgl01predictions[i] = 1
  } else {
    bestGAMfgl01predictions[i] = 0
  }
}
# predict(bestGAMfgl)
bestGAMfgl01predictions
sum(bestGAMfgl01predictions != fgl$type01)/nrow(fgl)
Nrep<-3 # number of replicates of CV

```

```

K<- 3 # Reducing the number of folds so that each fold has enough observations for the smoothers to work
n.models = 1 #number of different models to fit
n=nrow(fgl)
y<-fgl$type01
yhat=matrix(0,n,n.models)
misclass<-matrix(0,Nrep,n.models)
for (j in 1:Nrep) {
  Ind<-CVInd(n,K)
  for (k in 1:K) {
    train<-fgl[-Ind[[k]],] # Training data
    test<-fgl[Ind[[k]],1:9] # Test data for Predictors
    out<-gam(type01~s(RI) + s(Na) + s(Mg) + s(Al) + s(Si) + s(K) + s(Ca) + s(Ba) + s(Fe), data=train,
    # Numeric predictions
    numPredictions = predict(out,test,type = 'response')

    binaryPredictions = c(rep(0,nrow(test)))
    # "Binarizing" the predictions
    for (p in 1:length(numPredictions)){
      if (numPredictions[p]>=.5){
        binaryPredictions[p] = 1
      } else {
        binaryPredictions[p] = 0
      }
    }
    yhat[Ind[[k]],1] = binaryPredictions
  } #end of k loop
  misclass[j,] = sum(y != yhat)/n
} #end of j loop
misclassAve<- apply(misclass,2,mean) #averaged CV misclass rate
print(misclass)
print(misclassAve)

##### QUESTION 6 begins here #####
# loading data for question 6
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
# heart[,c(1:8)] = scale(heart[,c(1:8)]) # I THINK I DON'T NEED TO STANDARDIZE DATA HERE # Standardizing
heart <- as.data.frame(heart)
head(heart)
library(gbm)
# Fit gbm model
gbm1 <- gbm(log10_cost~., data=heart, var.monotone=rep(0,8), distribution="gaussian", n.trees=5000, shrinkage=.1)
bestM = gbm.perf(gbm1, method = 'cv')
bestM
gbm1$cv.error[60:70]
1-gbm1$cv.error[bestM]/var(heart$log10_cost)
summary(gbm1,n.trees=bestM)
par(mfrow = c(2,4))
plot(gbm1, i.var = 3, n.trees = bestM)
plot(gbm1, i.var = 7, n.trees = bestM)
plot(gbm1, i.var = 8, n.trees = bestM)
par(mfrow=c(1,1))
plot(gbm1, i.var = c(3,8), n.trees = bestM)

```

```

predVar = t(c(59,0,10,0,3,0,4,300))
predVar = as.data.frame((predVar))
names(predVar) = names(heart[,1:8])
predVar
prediction = predict(gbm1, predVar, n.trees = bestM)
prediction
10^3.446347
# gbm1 <- gbm(log10_cost~., data=heart, var.monotone=rep(0,8), distribution="gaussian", n.trees=5000, s
library(caret)
# tune.shrinkage = c(.001, .005, .01, .025, .05, .075, .01)
tune.shrinkage = seq(.001,.1,.005)
tune.M = seq(10, 1000, 1)
gbm.tuningParameters = expand.grid(tune.shrinkage, tune.M)
gbm.tuningParameters = data.frame(gbm.tuningParameters)
gbm.tuningParameters['n.minobsinnode'] = rep(10, nrow(gbm.tuningParameters))
gbm.tuningParameters['interaction.depth'] = rep(3,nrow(gbm.tuningParameters))
names(gbm.tuningParameters) = c('shrinkage', 'n.trees', 'n.minobsinnode', 'interaction.depth')
head(gbm.tuningParameters)
options = trainControl(method = 'repeatedcv', number = 10, repeats = 1, summaryFunction = defaultSummary)
gbm.cv = train(log10_cost ~ ., method = 'gbm', data = heart, trControl = options, tuneGrid = gbm.tuningParameters)
gbm.cv$bestTune
gbmCV <- gbm(log10_cost~., data=heart, var.monotone=rep(0,8), distribution="gaussian", n.trees=200, shrinkage=tune.shrinkage)
bestM = gbm.perf(gbmCV, method = 'cv')
bestM
1-gbmCV$cv.error[bestM]/var(heart$log10_cost)
1-gbmCV$cv.error[59]/var(heart$log10_cost)
##### QUESTION 7 begins here #####
# loading data for question 7
library(randomForest)
heart = read_excel('data.xls', sheet = 'Ischemic Heart Dis.')
heart['log10_cost'] = log10(heart[['cost']])
heart = heart[,-(1:2)] # Dropping the original cost column and the unique identifier
heart <- as.data.frame(heart)
head(heart)
# mtry: Number of variables randomly sampled as candidates at each split.
# It makes sense to try mtry = 3, since for regression the best is usually p/3, in this case 8/3
# ntree: number of trees to grow
myRF1 = randomForest(log10_cost~., data = heart, mtry = 3, ntree = 500)
myRF1
names(myRF1)
myRF1$rsq[480:500]
plot(myRF1)
myRF1$importance
par(mfrow = c(2,4))
for (i in c(1:8)) partialPlot(myRF1, pred.data=heart, x.var = names(heart)[i], xlab = names(heart)[i], ylab = names(heart)[1])

predVar = t(c(59,0,10,0,3,0,4,300))
predVar = as.data.frame((predVar))
names(predVar) = names(heart[,1:8])
predVar
predict(myRF1, predVar)
10^3.600718
myRF2 = randomForest(log10_cost~., data = heart, mtry = 1, ntree = 500)

```

```
myRF2
myRF3 = randomForest(log10_cost~., data = heart, mtry = 3, ntree = 500)
myRF3
```