

# Read Data

11/23/2021

```
library(foreign)
```

```
## Warning: package 'foreign' was built under R version 4.0.3
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.0.5
```

```
## -- Attaching packages ----- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5      v purrr   0.3.4
```

```
## v tibble  3.1.4      v dplyr   1.0.7
```

```
## v tidyr   1.1.3      v stringr 1.4.0
```

```
## v readr   2.0.1      v forcats 0.5.1
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

```
## Warning: package 'tibble' was built under R version 4.0.5
```

```
## Warning: package 'tidyr' was built under R version 4.0.4
```

```
## Warning: package 'readr' was built under R version 4.0.5
```

```
## Warning: package 'purrr' was built under R version 4.0.3
```

```
## Warning: package 'dplyr' was built under R version 4.0.5
```

```
## Warning: package 'stringr' was built under R version 4.0.4
```

```
## Warning: package 'forcats' was built under R version 4.0.4
```

```
## -- Conflicts ----- tidyverse_conflicts() --
```

```
## x dplyr::filter() masks stats::filter()
```

```
## x dplyr::lag()     masks stats::lag()
```

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.0.5
```

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.0.5
```

```
## Registered S3 method overwritten by 'tree':
```

```
##   method      from
```

```
##   print.tree cli
```

```
library(nnet)
```

```
## Warning: package 'nnet' was built under R version 4.0.5
```

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.0.5
```

```
## Loaded gbm 2.1.8
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.0.5
```

```
## randomForest 4.6-14
```

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

```
##
```

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

```
## The following object is masked from 'package:dplyr':
```

```
##
```

```
##   combine
```

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

```
##
```

```
##   margin
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.5
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.0.5
```

```
##
```

```
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
```

```
##
```

```
##   lift
```

```
library(ggplot2)
library(dplyr)
library(tidyr)
library(tidyverse)
library(patchwork)
```

```
## Warning: package 'patchwork' was built under R version 4.0.5
```

```
library(UBL)
```

```
## Warning: package 'UBL' was built under R version 4.0.5
```

```
## Loading required package: MBA
```

```
## Warning: package 'MBA' was built under R version 4.0.5
```

```
## Loading required package: gstat
```

```
## Warning: package 'gstat' was built under R version 4.0.5
```

```
## Loading required package: automap
```

```
## Warning: package 'automap' was built under R version 4.0.5
```

```
## Loading required package: sp
```

```
## Warning: package 'sp' was built under R version 4.0.5
```

```
library(scales)
```

```
## Warning: package 'scales' was built under R version 4.0.4
```

```
##
```

```
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
```

```
##
```

```
##      discard
```

```
## The following object is masked from 'package:readr':
```

```
##
```

```
##      col_factor
```

```
library(kableExtra)
```

```
## Warning: package 'kableExtra' was built under R version 4.0.5
```

```
##
## Attaching package: 'kableExtra'

## The following object is masked from 'package:dplyr':
##
##      group_rows

library(glmnet)

## Warning: package 'glmnet' was built under R version 4.0.5

## Loading required package: Matrix

## Warning: package 'Matrix' was built under R version 4.0.5

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##      expand, pack, unpack

## Loaded glmnet 4.1-2
```

```
sesame <- read.dta("sesame.dta")
sesame <- sesame %>%
  mutate(site=factor(site)) %>%
  mutate(bodyDiff = postbody - prebody,
         letDiff = postlet - prelet,
         formDiff = postform - preform,
         numbDiff = postnumb - prenumb,
         relatDiff = postrelat - prerelat,
         clasfDiff = postclasf - preclasf)
sesame.sd <- sesame%>%
  mutate(sd_pBod = scale(prebody, center = TRUE, scale = TRUE),
         sd_plet = scale(prelet, center = TRUE, scale = TRUE),
         sd_pform = scale(preform, center = TRUE, scale = TRUE),
         sd_pnumb = scale(prenumb, center = TRUE, scale = TRUE),
         sd_prelat = scale(prerelat, center = TRUE, scale = TRUE),
         sd_pclasf = scale(preclasf, center = TRUE, scale = TRUE),
         sd_peabody = scale(peabody, center = TRUE, scale = TRUE),
         sd_age = scale(age, center = TRUE, scale = TRUE),
         male = if_else(sex==1, 1, 0),
         female = if_else(sex==2, 1, 0))
```

## Exploratory Data Analysis

```
head(sesame)
```

##	rownames	id	site	sex	age	viewcat	setting	viewenc	prebody	prelet	preform
## 1		1	1	1	66	1	2	1	16	23	12
## 2		2	2	1	67	3	2	1	30	26	9
## 3		3	3	1	56	3	2	2	22	14	9
## 4		4	4	1	49	1	2	2	23	11	10
## 5		5	5	1	69	4	2	2	32	47	15
## 6		6	6	1	54	3	2	2	29	26	10
##	prenumb	prerelat	preclasf	postbody	postlet	postform	postnumb	postrelat			
## 1	40	14	20	18	30	14	44	14			
## 2	39	16	22	30	37	17	39	14			
## 3	9	9	8	21	46	15	40	9			
## 4	14	9	13	21	14	13	19	8			
## 5	51	17	22	32	63	18	54	14			
## 6	33	14	14	27	36	14	39	16			
##	postclasf	peabody	agecat	encour	_Isite_2	_Isite_3	_Isite_4	_Isite_5	regular		
## 1	23	62	1	1	0	0	0	0	0		
## 2	22	8	1	1	0	0	0	0	1		
## 3	19	32	1	0	0	0	0	0	1		
## 4	15	27	0	0	0	0	0	0	0		
## 5	21	71	1	0	0	0	0	0	1		
## 6	24	32	1	0	0	0	0	0	1		
##	bodyDiff	letDiff	formDiff	numbDiff	relatDiff	clasfDiff					
## 1	2	7	2	4	0	3					
## 2	0	11	8	0	-2	0					
## 3	-1	32	6	31	0	11					
## 4	-2	3	3	5	-1	2					
## 5	0	16	3	3	-3	-1					
## 6	-2	10	4	6	2	10					

## Variables:

The ID refers to a subject's identification number. The site refers to the age and background information of the child. A site value of 1 indicates a 3-5 year old disadvantaged child from the inner city. A site value of 2 represents a 4 year old advantaged child from the suburbs. A value of 3 represents an advantaged rural child. A site value of 4 indicates a disadvantaged rural child. Lastly, a value of 5 represents a disadvantaged Spanish speaking child. For the sex, a value of 1 indicates male, and a value of 2 indicates female. The age category is the child's age in months. The viewcat column is the frequency of viewing Sesame Street (1 = rarely, 2 = once/twice per week, 3 = 3-5 times a week, 4 = more than 5 times per week). The setting is where Sesame Street was viewed; a value of 1 indicates home and a value of 2 indicates school. The viewenc column refers to if the child was encouraged to watch or not (1 = child not encouraged, 2 = child encouraged). Encour is the same variable but with values 0 and 1, respectively. Regular is an indicator variable representing if a child is a regular viewer (0 = rarely watched, 1 = watched once per week or greater).

The prebody, prelet, preform, prenumb, prerelat, and preclasf columns all describe pretest scores on varying types of assessments (body parts, letters, forms, numbers, relational terms, and classification skills, respectively). The columns labelled postbody, postlet, postform, postnumb, postrelat, and postclasf are the children's respective posttest scores. Above, we created the following variables - bodyDiff, letDiff, formDiff, numbDiff, relatDiff, clasfDiff - to represent the difference in posttest scores and pretest scores for each child. Lastly, peabody represents a score of "mental age" for vocabulary maturity from the Peabody Picture Vocabulary Test.

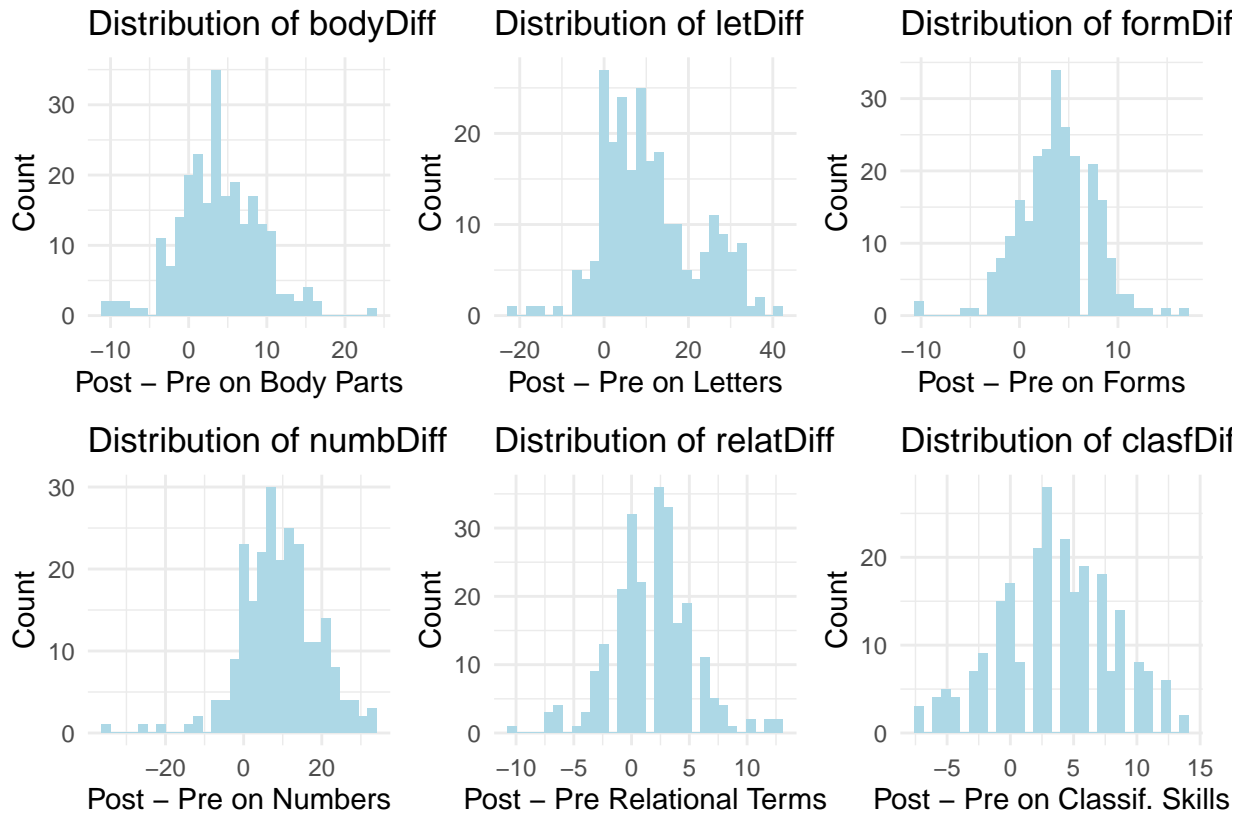
Our main focus will be on the new variables we created (bodyDiff, letDiff, formDiff, numbDiff, relatDiff, clasfDiff) and variables related to how often the children watch Sesame Street (namely, viewcat and regular). Lastly, we will look into the backgrounds of the children, including site, sex, and age.

## Distributions:

For the purposes of our analysis, we will first look at the distributions of bodyDiff, letDiff, formDiff, numbDiff, relatDiff, and clasfDiff.

*#want to visualize distributions of bodyDiff, letDiff, formDiff, numbDiff, relatDiff, clasfDiff*

```
bodyDiffplot <- ggplot(sesame, aes(x = bodyDiff)) +  
  geom_histogram(fill = "lightblue") +  
  labs(title = "Distribution of bodyDiff", x = "Post - Pre on Body Parts", y = "Count") +  
  theme_minimal()  
  
letDiffplot <- ggplot(sesame, aes(x = letDiff)) +  
  geom_histogram(fill = "lightblue") +  
  labs(title = "Distribution of letDiff", x = "Post - Pre on Letters", y = "Count") +  
  theme_minimal()  
  
formDiffplot <- ggplot(sesame, aes(x = formDiff)) +  
  geom_histogram(fill = "lightblue") +  
  labs(title = "Distribution of formDiff", x = "Post - Pre on Forms", y = "Count") +  
  theme_minimal()  
  
numbDiffplot <- ggplot(sesame, aes(x = numbDiff)) +  
  geom_histogram(fill = "lightblue") +  
  labs(title = "Distribution of numbDiff", x = "Post - Pre on Numbers", y = "Count") +  
  theme_minimal()  
  
relatDiffplot <- ggplot(sesame, aes(x = relatDiff)) +  
  geom_histogram(fill = "lightblue") +  
  labs(title = "Distribution of relatDiff", x = "Post - Pre Relational Terms", y = "Count") +  
  theme_minimal()  
  
clasfDiffplot <- ggplot(sesame, aes(x = clasfDiff)) +  
  geom_histogram(fill = "lightblue") +  
  labs(title = "Distribution of clasfDiff", x = "Post - Pre on Classif. Skills", y = "Count") +  
  theme_minimal()  
  
bodyDiffplot + letDiffplot + formDiffplot + numbDiffplot + relatDiffplot + clasfDiffplot
```



The six variables above were calculated by subtracting pre-test scores from post-test scores, so they are all numerical. The distributions of these six variables (bodyDiff, letDiff, formDiff, numbDiff, relatDiff, and clasfDiff) all appear to be roughly normal and unimodal. BodyDiff, letDiff, formDiff, relatDiff, and classDiff do not appear to have any obvious extreme outliers. Numbdiff, however, seems to be slightly left-skewed with outliers to the left -20. All of the six variables appear to have centers between 2 and 4.

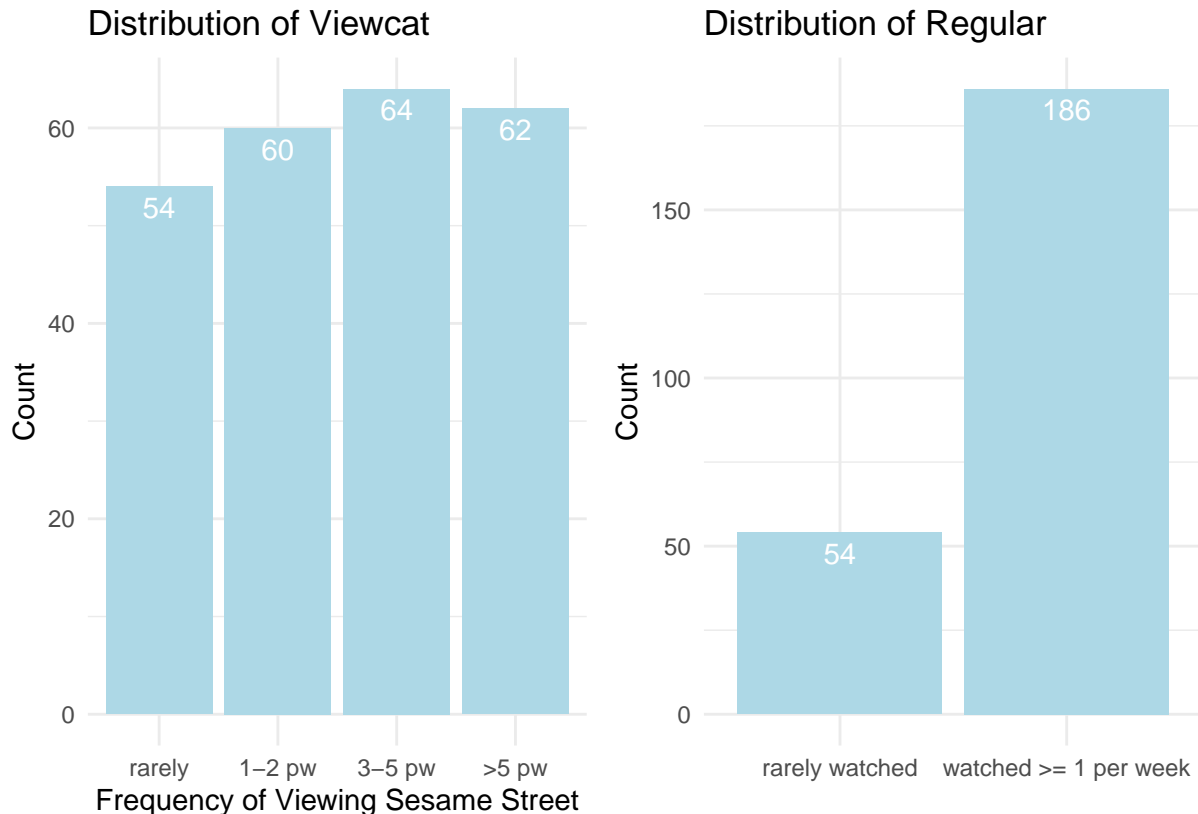
We will now examine the distributions of the variables related to how often children watch Sesame Street (namely, viewcat and regular).

```
# want to visualize distributions of viewcat and regular

viewcatplot <- ggplot(sesame, aes(x = factor(viewcat))) +
  geom_bar(fill = "lightblue") +
  labs(title = "Distribution of Viewcat", x = "Frequency of viewing Sesame Street", y = "Count") +
  scale_x_discrete("Frequency of Viewing Sesame Street", labels=c("rarely", "1-2 pw", "3-5 pw", ">5 pw")) +
  theme_minimal() +
  geom_text(aes(label = ..count..), stat = "count", vjust = 1.5, colour = "white")

regularplot <- ggplot(sesame, aes(x = factor(regular))) +
  geom_bar(fill = "lightblue") +
  labs(title = "Distribution of Regular", y = "Count") +
  scale_x_discrete(labels=c("rarely watched", "watched >= 1 per week")) +
  theme_minimal() +
  theme(axis.title.x = element_blank()) +
  geom_text(aes(label = ..count..), stat = "count", vjust = 1.5, colour = "white")
```

```
viewcatplot + regularplot
```



Both of these variables are categorical. On the left, viewcat appears to have a roughly uniform distribution, with “rarely” having the least amount of children and 3-5 times per week having the most (the range is only 10 children, so all of the bars are relatively close in height). For the variable regular, the category “watched once per week or greater” has far more observations than “rarely watched.” The former category has more than triple the amount of the latter. We will be aware of this disparity in our analysis and continue with caution towards potential bias.

Lastly, we want to examine the distributions of site, sex, and age, all variables that relate to a child’s background.

```
# want to visualize distributions of site, sex, and age

siteplot <- ggplot(sesame, aes(x = factor(site))) +
  geom_bar(fill = "lightblue") +
  labs(title = "Distribution of Site", y = "Count") +
  scale_x_discrete(labels=c("3-5, disadv., inner city", "4, adv., suburb", "adv., rural", "disadv., rural")) +
  theme_minimal() +
  theme(axis.title.x = element_blank()) +
  geom_text(aes(label = ..count..), stat = "count", vjust = 1.5, colour = "white")

sexplot <- ggplot(sesame, aes(x = factor(sex))) +
  geom_bar(fill = "lightblue") +
  labs(title = "Distribution of Sex", y = "Count") +
  scale_x_discrete(labels=c("Male", "Female")) +
```



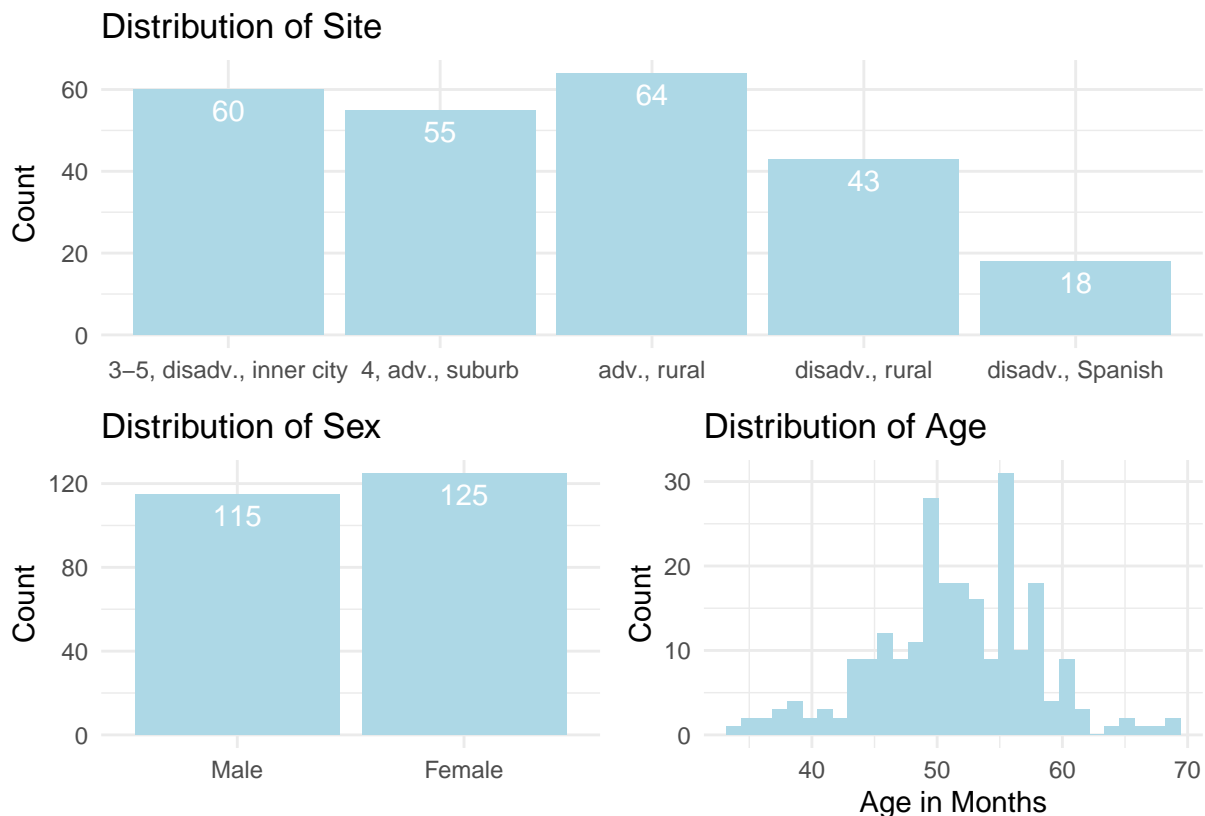
```

theme_minimal() +
  theme(axis.title.x = element_blank()) +
  geom_text(aes(label = ..count..), stat = "count", vjust = 1.5, colour = "white")

ageplot <- ggplot(sesame, aes(x = age)) +
  geom_histogram(fill = "lightblue") +
  labs(title = "Distribution of Age", x = "Age in Months", y = "Count") +
  theme_minimal()

siteplot / (sexplot + ageplot)

```



Site and Sex are both categorical variables. Distribution of Site has four categories with roughly the same amount of children (ranging from 43 to 64), but one category with far fewer observations (disadvantaged Spanish-speaking). This category has less than half of the observations as the next smallest category, which is a relatively large disparity. We will continue our analysis with caution towards this bias in the data. The distribution of sex is very even - the male category has 115 observations, while the female category has 125 observations. Age is a numerical variable that appears to be normal and bimodal, with two peaks around 50 and 56. There do not appear to be any extreme outliers in the distribution of age.

**Q.1 Prediction Question:** Can we use linear regression to predict the change in a child's test scores that occur after watching Sesame street (or in some instances, not watching Sesame street)?

## Linear Regression Models

Here, I am fitting 6 linear regression models. Each of the models predicts a different difference in test score.

```
sesame.q1 <- sesame

sesame.q1$site <- as.factor(sesame.q1$site)
sesame.q1$sex <- as.factor(sesame.q1$sex)
sesame.q1$viewcat <- as.factor(sesame.q1$viewcat)
sesame.q1$setting <- as.factor(sesame.q1$setting)
sesame.q1$viewenc <- as.factor(sesame.q1$viewenc)

# Scaling Variables

sesame.q1$bodyDiff <- rescale(sesame.q1$bodyDiff, to = c(0, 30))
sesame.q1$letDiff <- rescale(sesame.q1$letDiff, to = c(0, 30))
sesame.q1$formDiff <- rescale(sesame.q1$formDiff, to = c(0, 30))
sesame.q1$numbDiff <- rescale(sesame.q1$numbDiff, to = c(0, 30))
sesame.q1$relatDiff <- rescale(sesame.q1$relatDiff, to = c(0, 30))
sesame.q1$clasfDiff <- rescale(sesame.q1$clasfDiff, to = c(0, 30))

# Test-Train Split

set.seed(1)
train <- sample(1:nrow(sesame.q1), nrow(sesame.q1)*0.7)

training = sesame.q1[train,]
testing = sesame.q1[-train, ]
```

Before creating these models, we first factored the following variables to encode them as categoricals: `site`, `sex`, `viewcat`, `setting`, `viewenc`. One problem that we envisioned when evaluating and comparing the different models is that the tests are scored on different scales. For example, the scores for the test on knowledge of body parts (noted by `bodyDiff`) range from 0-32, while those of the test on letters (noted by `letDiff`) range from 0-58. To be able to aptly compare the mean squared error (MSE) between models, we also decided to convert each response variable to the same range. More specifically, we scaled each variable to the arbitrary range [0, 30]. Lastly, we randomly split the data between testing and training, using 70% of the data for training and 30% of the data for testing.

```
lin.mod1.full <- lm(bodyDiff ~ (site + sex + age + viewcat + setting + viewenc), data = training)
summary(lin.mod1.full)

##
## Call:
## lm(formula = bodyDiff ~ (site + sex + age + viewcat + setting +
##      viewenc), data = training)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -11.3707 -2.7927 0.0525 2.5545 13.1411
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 16.42668   3.06383   5.361 2.92e-07 ***
## site2       0.58666   0.99737   0.588 0.55724
## site3       2.44827   0.96820   2.529 0.01244 *
## site4       2.96605   1.10488   2.685 0.00805 **
## site5       3.29854   1.45590   2.266 0.02485 *
## sex2        -1.00041   0.66732  -1.499 0.13586
## age         -0.08726   0.05558  -1.570 0.11845
## viewcat2    -0.60564   1.12908  -0.536 0.59244
## viewcat3     1.30568   1.09680   1.190 0.23568
## viewcat4     2.18215   1.12555   1.939 0.05434 .
## setting2    -1.02953   0.77713  -1.325 0.18718
## viewenc2    -0.37832   0.84111  -0.450 0.65349
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.253 on 156 degrees of freedom
## Multiple R-squared:  0.1585, Adjusted R-squared:  0.09917
## F-statistic: 2.671 on 11 and 156 DF, p-value: 0.00362
```

```
AIC(lin.mod1.full)
```

```
## [1] 976.6991
```

```
yhat <- predict(lin.mod1.full, newdata = testing)
y.test <- testing[, "bodyDiff"]
```

```
# Test MSE
mean((yhat-y.test)^2)
```

```
## [1] 24.01045
```

```
# Split data between x and y
x.train <- model.matrix(bodyDiff~site + sex + age + viewcat + setting + viewenc, training)[,-1]
y.train <- training$bodyDiff

x.test <- model.matrix(bodyDiff~site + sex + age + viewcat + setting + viewenc, testing)[,-1]
y.test <- testing$bodyDiff

# set seed
set.seed(1)

# cross validation for lambda
cv.out <- cv.glmnet(x.train, y.train, alpha = 0) # setting alpha = 0 indicates ridge regression

# optimal lambda value
best.lam <- cv.out$lambda.min

# ridge regression model with optimal lambda
```

```

ridge.mod1.full <- glmnet(x.train, y.train, alpha = 0, lambda = best.lam)

# calculate predictions
ridge.pred <- predict(ridge.mod1.full, s = best.lam, newx = x.test)

# MSE calculation
mean((ridge.pred - y.test)^2)

## [1] 21.60675

lin.mod2.full <- lm(letDiff ~ (site + sex + age + viewcat + setting + viewenc), data = training)
summary(lin.mod2.full)

##
## Call:
## lm(formula = letDiff ~ (site + sex + age + viewcat + setting +
##      viewenc), data = training)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.2823  -2.9776  -0.3663   2.9275  11.1426
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.52353    3.32595   3.164  0.00187 **
## site2        3.08586    1.08270   2.850  0.00496 **
## site3       -2.85941    1.05103  -2.721  0.00726 **
## site4       -1.07861    1.19940  -0.899  0.36989
## site5       -0.38501    1.58046  -0.244  0.80786
## sex2         0.37911    0.72441   0.523  0.60148
## age          0.05050    0.06034   0.837  0.40390
## viewcat2     1.59616    1.22568   1.302  0.19474
## viewcat3     4.85135    1.19064   4.075 7.32e-05 ***
## viewcat4     4.77123    1.22184   3.905  0.00014 ***
## setting2     0.70346    0.84362   0.834  0.40563
## viewenc2    -1.56705    0.91307  -1.716  0.08810 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.617 on 156 degrees of freedom
## Multiple R-squared:  0.37, Adjusted R-squared:  0.3256
## F-statistic: 8.33 on 11 and 156 DF, p-value: 2.043e-11

AIC(lin.mod2.full)

## [1] 1004.281

yhat <- predict(lin.mod2.full, newdata = testing)
y.test <- testing[, "letDiff"]

# Test MSE
mean((yhat-y.test)^2)

```

```
## [1] 14.31277
```

```
# Split data between x and y
x.train <- model.matrix(letDiff~site + sex + age + viewcat + setting + viewenc, training)[-1]
y.train <- training$letDiff

x.test <- model.matrix(letDiff~site + sex + age + viewcat + setting + viewenc, testing)[-1]
y.test <- testing$letDiff

# set seed
set.seed(1)

# cross validation for lambda
cv.out <- cv.glmnet(x.train, y.train, alpha = 0) # setting alpha = 0 indicates ridge regression

# optimal lambda value
best.lam <- cv.out$lambda.min

# ridge regression model with optimal lambda
ridge.mod2.full <- glmnet(x.train, y.train, alpha = 0, lambda = best.lam)

# calculate predictions
ridge.pred <- predict(ridge.mod2.full, s = best.lam, newx = x.test)

# MSE calculation
mean((ridge.pred - y.test)^2)
```

```
## [1] 14.19572
```

```
lin.mod3.full <- lm(formDiff ~ (site + sex + age + viewcat + setting + viewenc), data = training)
summary(lin.mod3.full)
```

```
##
## Call:
## lm(formula = formDiff ~ (site + sex + age + viewcat + setting +
##      viewenc), data = training)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.3637  -2.5344   0.2658   2.7054  14.1942
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  16.04193    3.16388   5.070 1.11e-06 ***
## site2         0.97029    1.02994   0.942  0.3476
## site3         0.73495    0.99982   0.735  0.4634
## site4         0.92962    1.14096   0.815  0.4164
## site5         1.00499    1.50344   0.668  0.5048
## sex2          0.09439    0.68911   0.137  0.8912
## age          -0.05234    0.05740  -0.912  0.3632
## viewcat2       0.99101    1.16595   0.850  0.3966
## viewcat3       1.65614    1.13262   1.462  0.1457
## viewcat4       2.47768    1.16230   2.132  0.0346 *
```

```
## setting2      -0.08994      0.80251  -0.112    0.9109
## viewenc2      -0.42935      0.86858  -0.494    0.6218
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.392 on 156 degrees of freedom
## Multiple R-squared:  0.06302,    Adjusted R-squared:  -0.003048
## F-statistic: 0.9539 on 11 and 156 DF,  p-value: 0.4909
```

```
AIC(lin.mod3.full)
```

```
## [1] 987.4956
```

```
yhat <- predict(lin.mod3.full, newdata = testing)
y.test <- testing[, "formDiff"]
```

```
# Test MSE
mean((yhat-y.test)^2)
```

```
## [1] 13.25639
```

```
# Split data between x and y
x.train <- model.matrix(formDiff~site + sex + age + viewcat + setting + viewenc, training)[,-1]
y.train <- training$formDiff
```

```
x.test <- model.matrix(formDiff~site + sex + age + viewcat + setting + viewenc, testing)[,-1]
y.test <- testing$formDiff
```

```
# set seed
set.seed(1)
```

```
# cross validation for lambda
cv.out <- cv.glmnet(x.train, y.train, alpha = 0) # setting alpha = 0 indicates ridge regression
```

```
# optimal lambda value
best.lam <- cv.out$lambda.min
```

```
# ridge regression model with optimal lambda
ridge.mod3.full <- glmnet(x.train, y.train, alpha = 0, lambda = best.lam)
```

```
# calculate predictions
ridge.pred <- predict(ridge.mod3.full, s = best.lam, newx = x.test)
```

```
# MSE calculation
mean((ridge.pred - y.test)^2)
```

```
## [1] 12.82502
```

```
lin.mod4.full <- lm(numDiff ~ (site + sex + age + viewcat + setting + viewenc), data = training)
summary(lin.mod4.full)
```

```
##
## Call:
## lm(formula = numbDiff ~ (site + sex + age + viewcat + setting +
##      viewenc), data = training)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -19.6433  -2.6010   0.1375   2.3541  10.3247
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 15.388591   3.084422   4.989 1.6e-06 ***
## site2       2.561427   1.004073   2.551 0.01170 *
## site3       0.768616   0.974707   0.789 0.43157
## site4       1.151651   1.112304   1.035 0.30210
## site5       1.946294   1.465685   1.328 0.18615
## sex2        0.013992   0.671801   0.021 0.98341
## age         0.009505   0.055956   0.170 0.86534
## viewcat2    1.841715   1.136668   1.620 0.10719
## viewcat3    3.199614   1.104174   2.898 0.00430 **
## viewcat4    3.636674   1.133111   3.209 0.00161 **
## setting2    0.596707   0.782354   0.763 0.44679
## viewenc2    0.589407   0.846765   0.696 0.48742
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.281 on 156 degrees of freedom
## Multiple R-squared:  0.139, Adjusted R-squared:  0.07825
## F-statistic: 2.289 on 11 and 156 DF, p-value: 0.01273
```

```
AIC(lin.mod4.full)
```

```
## [1] 978.9495
```

```
yhat <- predict(lin.mod4.full, newdata = testing)
y.test <- testing[, "numbDiff"]
```

```
# Test MSE
mean((yhat-y.test)^2)
```

```
## [1] 15.5095
```

```
# Split data between x and y
x.train <- model.matrix(numbDiff~site + sex + age + viewcat + setting + viewenc, training)[,-1]
y.train <- training$numbDiff

x.test <- model.matrix(numbDiff~site + sex + age + viewcat + setting + viewenc, testing)[,-1]
y.test <- testing$numbDiff

# set seed
set.seed(1)
```

```

# cross validation for lambda
cv.out <- cv.glmnet(x.train, y.train, alpha = 0) # setting alpha = 0 indicates ridge regression

# optimal lambda value
best.lam <- cv.out$lambda.min

# ridge regression model with optimal lambda
ridge.mod4.full <- glmnet(x.train, y.train, alpha = 0, lambda = best.lam)

# calculate predictions
ridge.pred <- predict(ridge.mod4.full, s = best.lam, newx = x.test)

# MSE calculation
mean((ridge.pred - y.test)^2)

## [1] 14.62731

lin.mod5.full <- lm(relatDiff ~ (site + sex + age + viewcat + setting + viewenc), data = training)
summary(lin.mod5.full)

##
## Call:
## lm(formula = relatDiff ~ (site + sex + age + viewcat + setting +
##      viewenc), data = training)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.9246  -2.7114   0.3439   2.2185  14.4415
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  18.48969    3.17203   5.829 3.1e-08 ***
## site2        -0.05900    1.03259  -0.057  0.9545
## site3         1.66242    1.00239   1.658  0.0992 .
## site4         2.41955    1.14390   2.115  0.0360 *
## site5        -0.40561    1.50731  -0.269  0.7882
## sex2          0.31932    0.69088   0.462  0.6446
## age          -0.09729    0.05755  -1.691  0.0929 .
## viewcat2      1.05190    1.16895   0.900  0.3696
## viewcat3      0.96616    1.13554   0.851  0.3962
## viewcat4      2.95954    1.16529   2.540  0.0121 *
## setting2     -0.60666    0.80457  -0.754  0.4520
## viewenc2      0.19462    0.87082   0.223  0.8234
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.403 on 156 degrees of freedom
## Multiple R-squared:  0.1072, Adjusted R-squared:  0.04421
## F-statistic: 1.702 on 11 and 156 DF, p-value: 0.07739

AIC(lin.mod5.full)

## [1] 988.3598

```



```

yhat <- predict(lin.mod5.full, newdata = testing)
y.test <- testing[, "relatDiff"]

# Test MSE
mean((yhat-y.test)^2)

## [1] 18.94131

# Split data between x and y
x.train <- model.matrix(relatDiff~site + sex + age + viewcat + setting + viewenc, training)[,-1]
y.train <- training$relatDiff

x.test <- model.matrix(relatDiff~site + sex + age + viewcat + setting + viewenc, testing)[,-1]
y.test <- testing$relatDiff

# set seed
set.seed(1)

# cross validation for lambda
cv.out <- cv.glmnet(x.train, y.train, alpha = 0) # setting alpha = 0 indicates ridge regression

# optimal lambda value
best.lam <- cv.out$lambda.min

# ridge regression model with optimal lambda
ridge.mod5.full <- glmnet(x.train, y.train, alpha = 0, lambda = best.lam)

# calculate predictions
ridge.pred <- predict(ridge.mod5.full, s = best.lam, newx = x.test)

# MSE calculation
mean((ridge.pred - y.test)^2)

## [1] 19.86283

lin.mod6.full <- lm(clasfDiff ~ (site + sex + age + viewcat + setting + viewenc), data = training)
summary(lin.mod6.full)

##
## Call:
## lm(formula = clasfDiff ~ (site + sex + age + viewcat + setting +
##      viewenc), data = training)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.1077  -4.0442   0.2665   3.8807  14.5260
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.951998   4.455915   2.458  0.0151 *
## site2       1.799749   1.450536   1.241  0.2166
## site3       0.420798   1.408112   0.299  0.7655

```

```
## site4      2.040728   1.606891   1.270   0.2060
## site5      1.497017   2.117404   0.707   0.4806
## sex2       1.500233   0.970519   1.546   0.1242
## age       -0.008915   0.080838  -0.110   0.9123
## viewcat2    3.041609   1.642090   1.852   0.0659 .
## viewcat3    3.072479   1.595147   1.926   0.0559 .
## viewcat4    4.060404   1.636951   2.480   0.0142 *
## setting2    0.477618   1.130229   0.423   0.6732
## viewenc2   -1.088979   1.223281  -0.890   0.3747
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.185 on 156 degrees of freedom
## Multiple R-squared:  0.09426,    Adjusted R-squared:  0.03039
## F-statistic: 1.476 on 11 and 156 DF,  p-value: 0.1455
```

```
AIC(lin.mod6.full)
```

```
## [1] 1102.553
```

```
yhat <- predict(lin.mod6.full, newdata = testing)
y.test <- testing[, "clasfDiff"]
```

```
# Test MSE
mean((yhat-y.test)^2)
```

```
## [1] 48.398
```

```
# Split data between x and y
x.train <- model.matrix(clasfDiff~site + sex + age + viewcat + setting + viewenc, training)[,-1]
y.train <- training$clasfDiff

x.test <- model.matrix(clasfDiff~site + sex + age + viewcat + setting + viewenc, testing)[,-1]
y.test <- testing$clasfDiff

# set seed
set.seed(1)

# cross validation for lambda
cv.out <- cv.glmnet(x.train, y.train, alpha = 0) # setting alpha = 0 indicates ridge regression

# optimal lambda value
best.lam <- cv.out$lambda.min

# ridge regression model with optimal lambda
ridge.mod6.full <- glmnet(x.train, y.train, alpha = 0, lambda = best.lam)

# calculate predictions
ridge.pred <- predict(ridge.mod6.full, s = best.lam, newx = x.test)

# MSE calculation
mean((ridge.pred - y.test)^2)
```

```
## [1] 44.26185
```

## Regression Tree Models

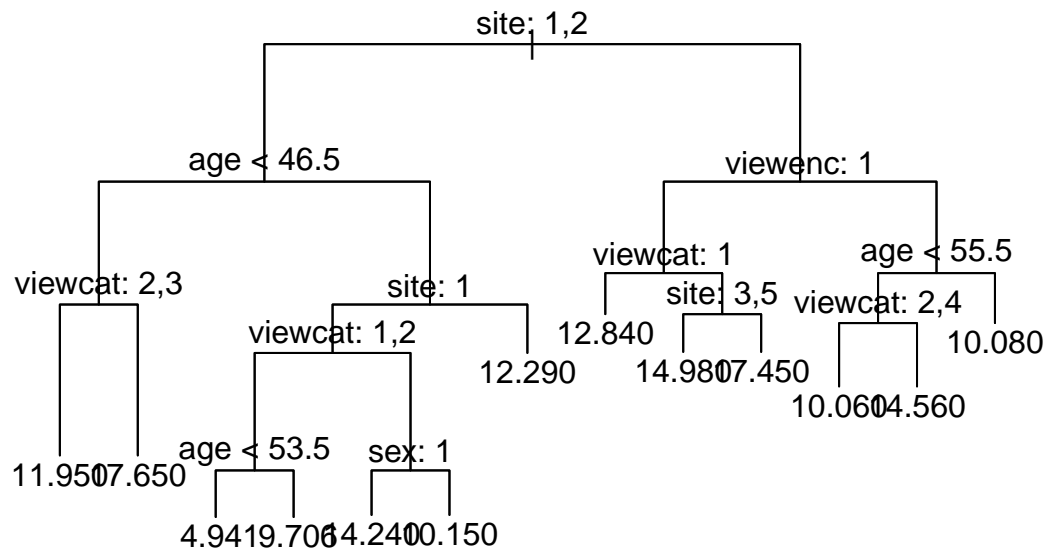
### Model 1

```
set.seed(1)

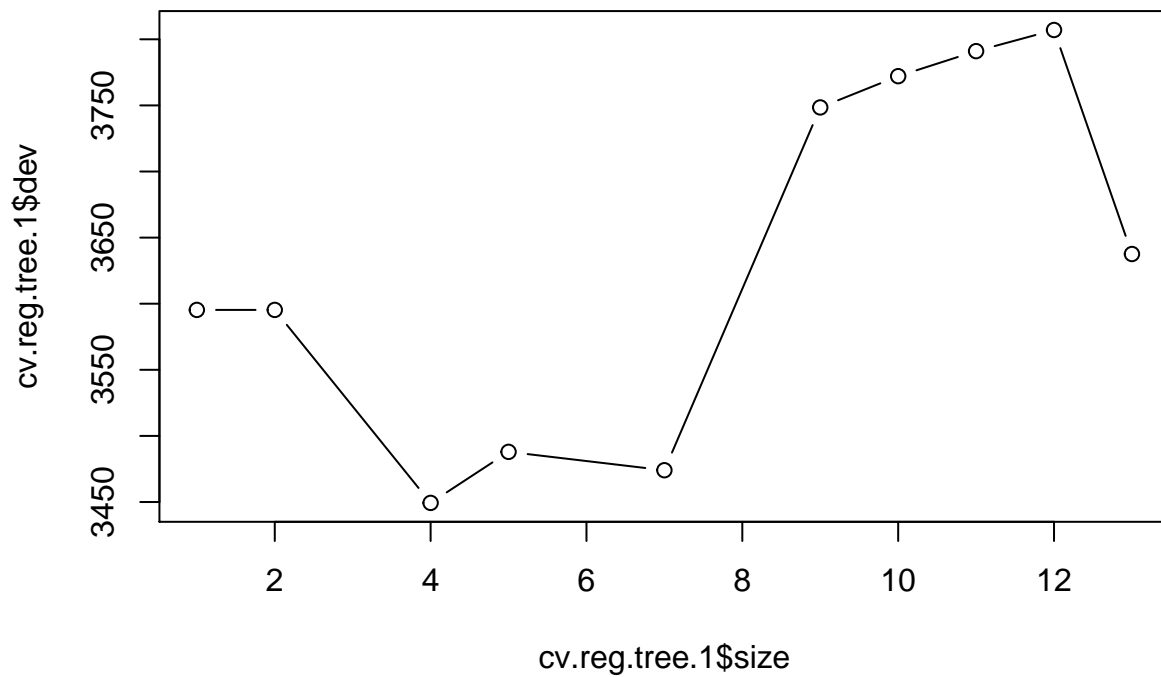
reg.tree.1 <- tree(bodyDiff ~ site + sex + age + viewcat + setting + viewenc, sesame.q1, subset = train)
summary(reg.tree.1)

##
## Regression tree:
## tree(formula = bodyDiff ~ site + sex + age + viewcat + setting +
##       viewenc, data = sesame.q1, subset = train)
## Variables actually used in tree construction:
## [1] "site"    "age"     "viewcat" "sex"     "viewenc"
## Number of terminal nodes: 13
## Residual mean deviance: 14.16 = 2194 / 155
## Distribution of residuals:
##      Min.   1st Qu.   Median     Mean   3rd Qu.    Max.
## -11.08000 -2.45100 -0.06303  0.00000  2.16900 12.55000

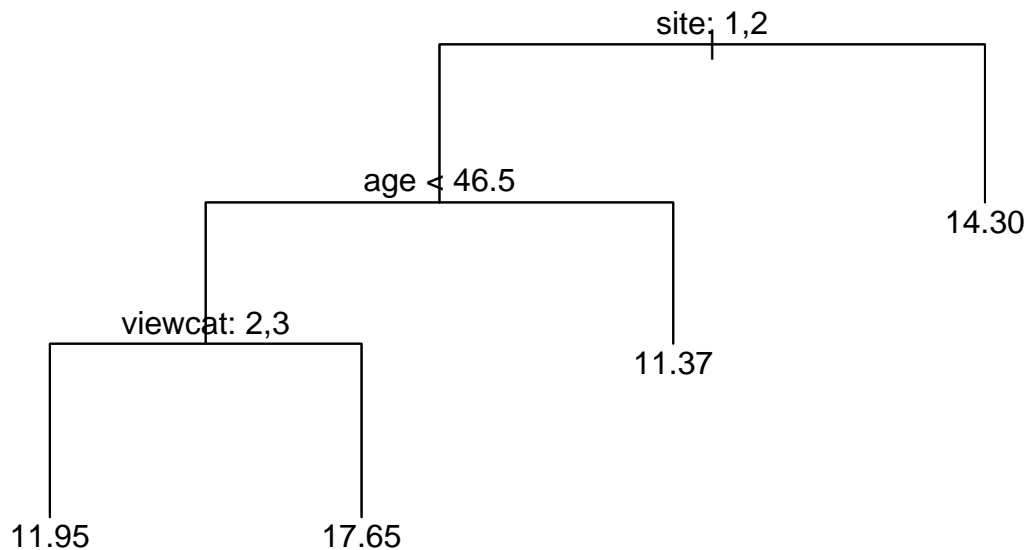
plot(reg.tree.1)
text(reg.tree.1, pretty = 0)
```



```
cv.reg.tree.1 <- cv.tree(reg.tree.1)
plot(cv.reg.tree.1$size, cv.reg.tree.1$dev, type = "b")
```



```
prune.reg.tree.1 <- prune.tree(reg.tree.1, best = 4)
plot(prune.reg.tree.1)
text(prune.reg.tree.1, pretty = 0)
```



```
yhat <- predict(prune.reg.tree.1, newdata = testing)
y.test <- testing[, "bodyDiff"]

# Test MSE
mean((yhat-y.test)^2)
```

```
## [1] 20.60159
```

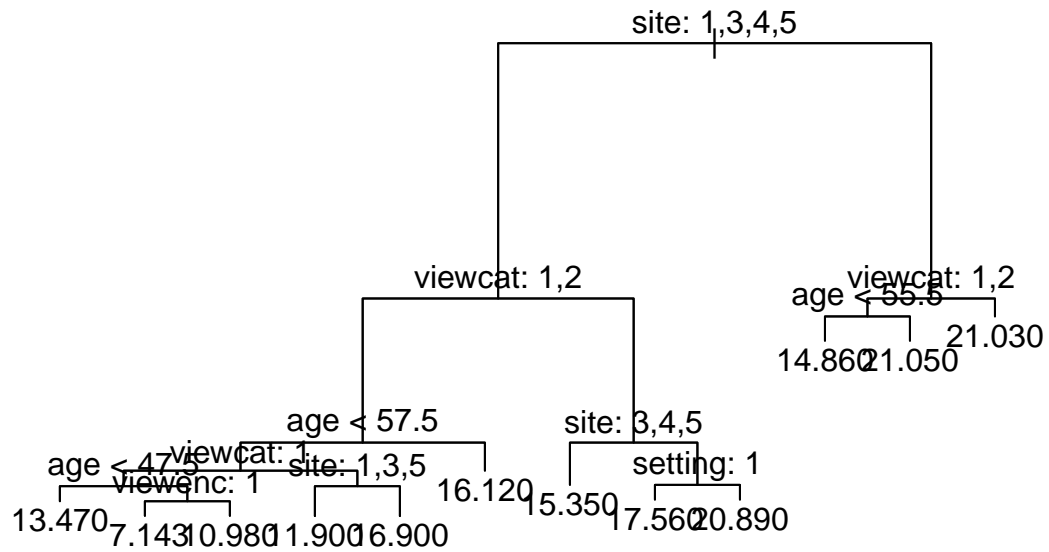
## Model 2

```
set.seed(1)

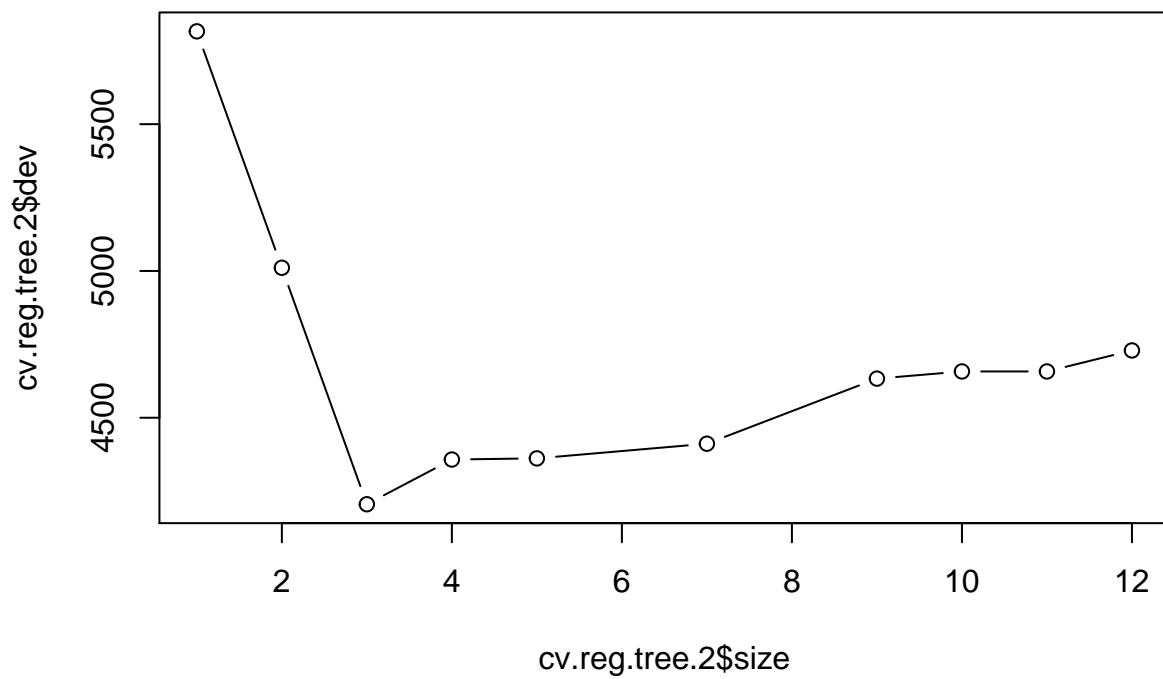
reg.tree.2 <- tree(letDiff ~ site + sex + age + viewcat + setting + viewenc, sesame.q1, subset = train)
summary(reg.tree.2)

##
## Regression tree:
## tree(formula = letDiff ~ site + sex + age + viewcat + setting +
##       viewenc, data = sesame.q1, subset = train)
## Variables actually used in tree construction:
## [1] "site"      "viewcat"   "age"       "viewenc"   "setting"
## Number of terminal nodes: 12
## Residual mean deviance: 18.63 = 2906 / 156
```

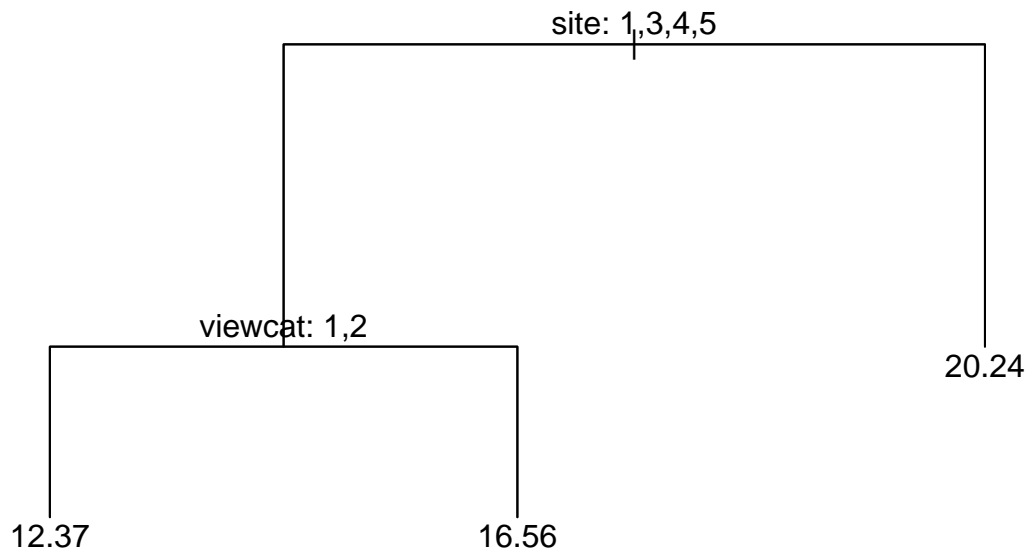
```
plot(reg.tree.2)
text(reg.tree.2, pretty = 0)
```



22



```
prune.reg.tree.2 <- prune.tree(reg.tree.2, best = 3)
plot(prune.reg.tree.2)
text(prune.reg.tree.2, pretty = 0)
```



```

yhat <- predict(prune.reg.tree.2, newdata = testing)
y.test <- testing[, "letDiff"]

# Test MSE
mean((yhat-y.test)^2)

```

```
## [1] 15.4124
```

### Model 3

```

set.seed(1)

reg.tree.3 <- tree(formDiff ~ site + sex + age + viewcat + setting + viewenc, sesame.q1, subset = train)
summary(reg.tree.3)

```

```

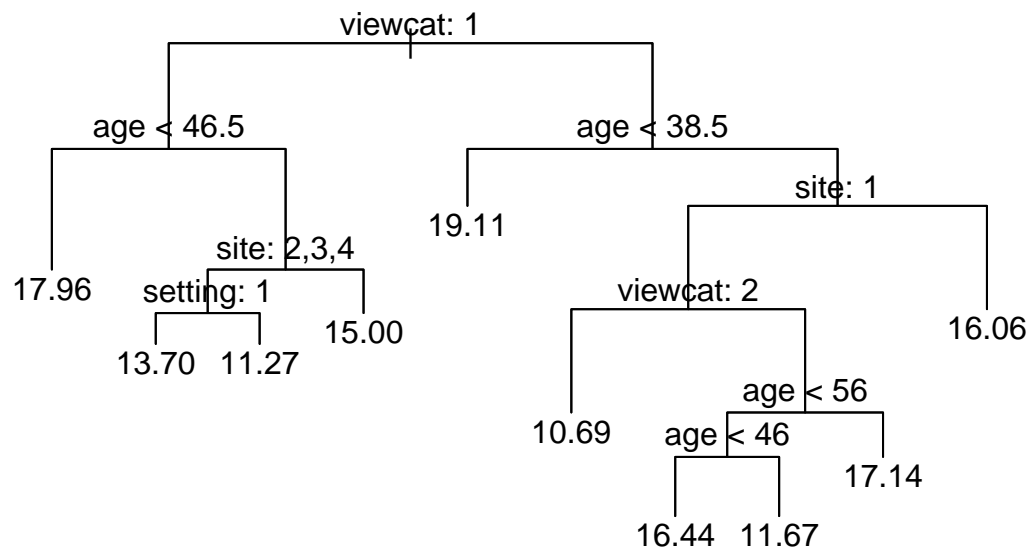
##
## Regression tree:
## tree(formula = formDiff ~ site + sex + age + viewcat + setting +
##       viewenc, data = sesame.q1, subset = train)
## Variables actually used in tree construction:
## [1] "viewcat" "age"      "site"      "setting"
## Number of terminal nodes: 10
## Residual mean deviance: 15.88 = 2509 / 158

```

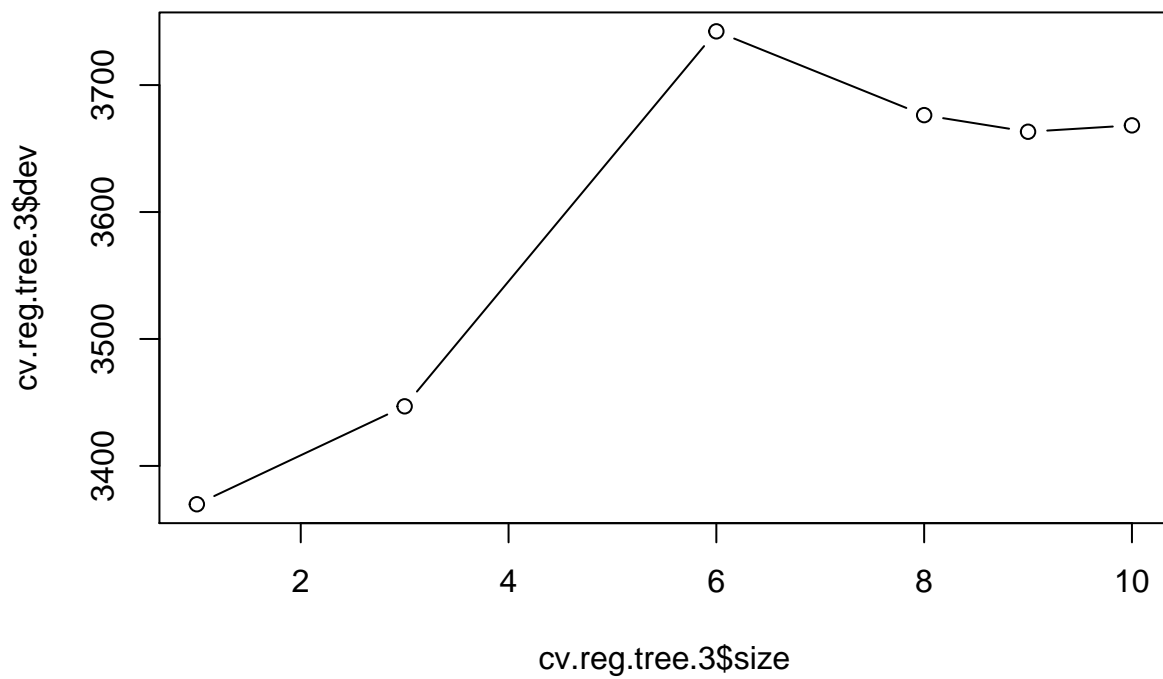


```
## Distribution of residuals:
##   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -11.610 -1.667   0.129   0.000   2.159   13.940
```

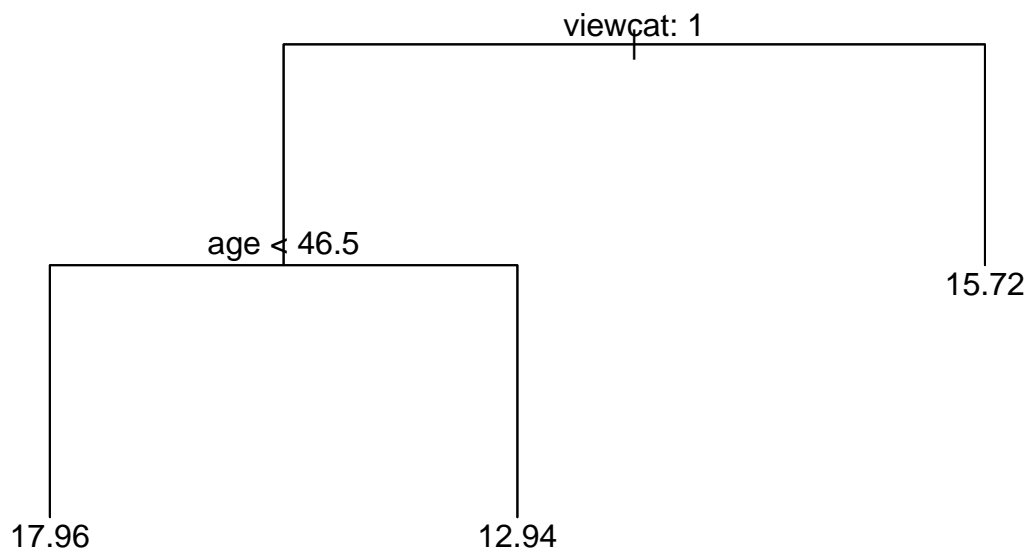
```
plot(reg.tree.3)
text(reg.tree.3, pretty = 0)
```



```
cv.reg.tree.3 <- cv.tree(reg.tree.3)
plot(cv.reg.tree.3$size, cv.reg.tree.3$dev, type = "b")
```



```
prune.reg.tree.3 <- prune.tree(reg.tree.3, best = 2)
plot(prune.reg.tree.3)
text(prune.reg.tree.3, pretty = 0)
```



```
yhat <- predict(prune.reg.tree.3, newdata = testing)
y.test <- testing[, "formDiff"]
```

```
# Test MSE
mean((yhat-y.test)^2)
```

```
## [1] 14.92273
```

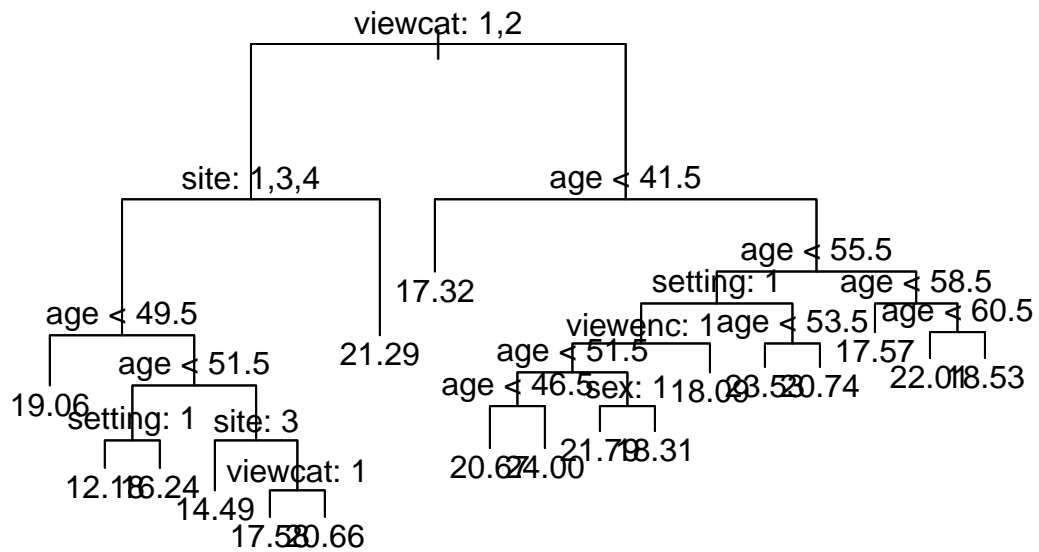
## Model 4

```
set.seed(1)
```

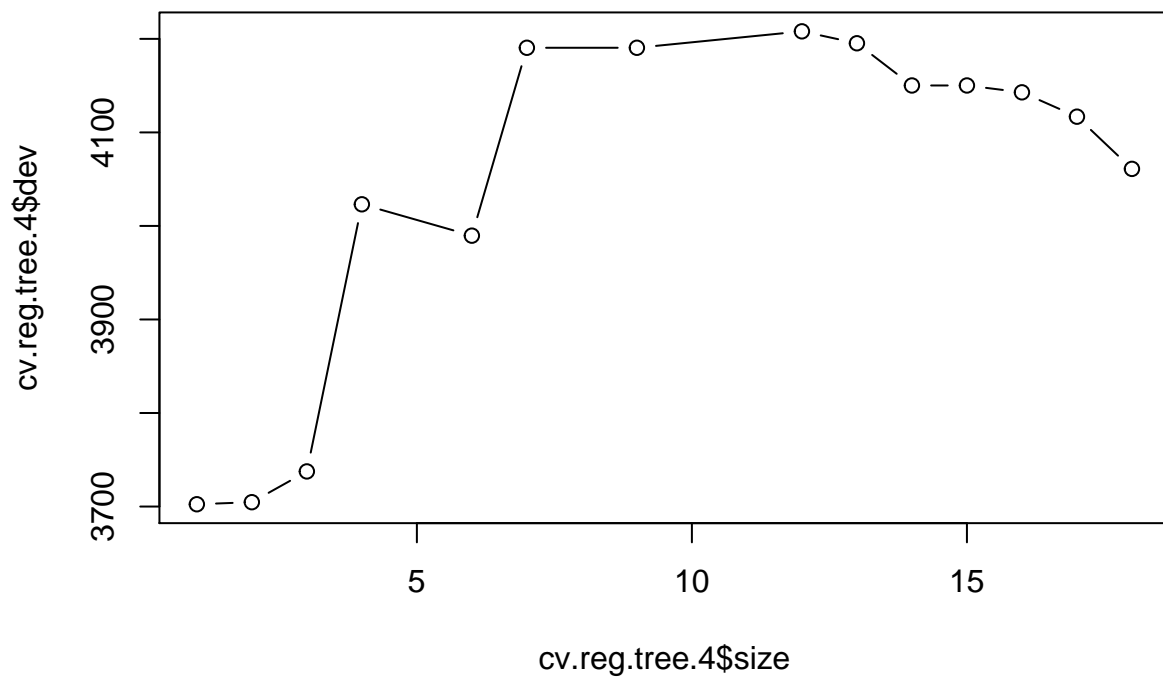
```
reg.tree.4 <- tree(numDiff ~ site + sex + age + viewcat + setting + viewenc, sesame.q1, subset = train)
summary(reg.tree.4)
```

```
##
## Regression tree:
## tree(formula = numDiff ~ site + sex + age + viewcat + setting +
##       viewenc, data = sesame.q1, subset = train)
## Number of terminal nodes: 18
## Residual mean deviance: 13.64 = 2046 / 150
## Distribution of residuals:
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -18.0900 -2.1210   0.2647   0.0000  2.1180  11.4700
```

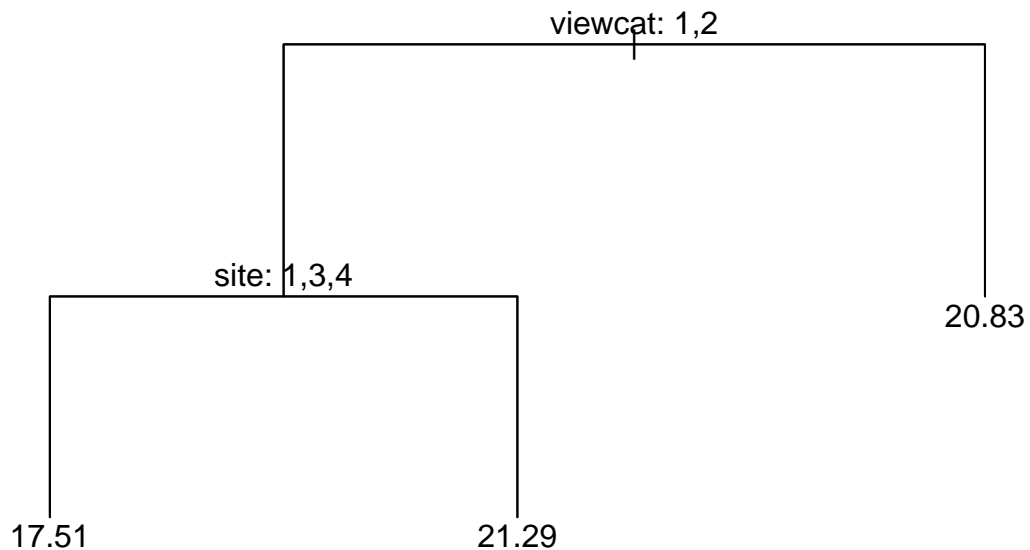
```
plot(reg.tree.4)
text(reg.tree.4, pretty = 0)
```



```
cv.reg.tree.4 <- cv.tree(reg.tree.4)
plot(cv.reg.tree.4$size, cv.reg.tree.4$dev, type = "b")
```



```
prune.reg.tree.4 <- prune.tree(reg.tree.4, best = 3)
plot(prune.reg.tree.4)
text(prune.reg.tree.4, pretty = 0)
```



```

yhat <- predict(prune.reg.tree.4, newdata = testing)
y.test <- testing[, "numbDiff"]

# Test MSE
mean((yhat-y.test)^2)

```

```
## [1] 15.90771
```

## Model 5

```

set.seed(1)

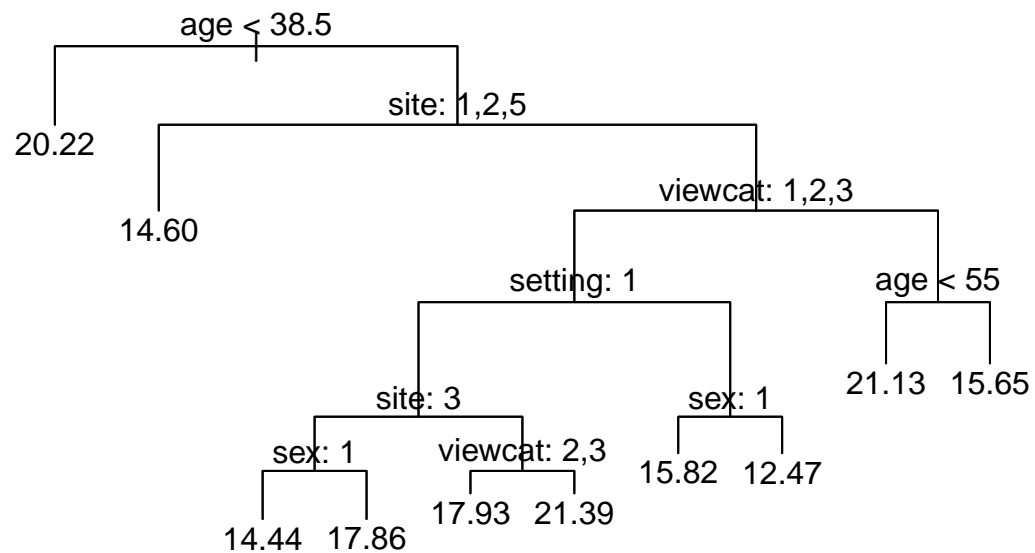
reg.tree.5 <- tree(relatDiff ~ site + sex + age + viewcat + setting + viewenc, sesame.q1, subset = train)
summary(reg.tree.5)

##
## Regression tree:
## tree(formula = relatDiff ~ site + sex + age + viewcat + setting +
##       viewenc, data = sesame.q1, subset = train)
## Variables actually used in tree construction:
## [1] "age"      "site"     "viewcat"  "setting"  "sex"
## Number of terminal nodes: 10
## Residual mean deviance: 15.3 = 2418 / 158

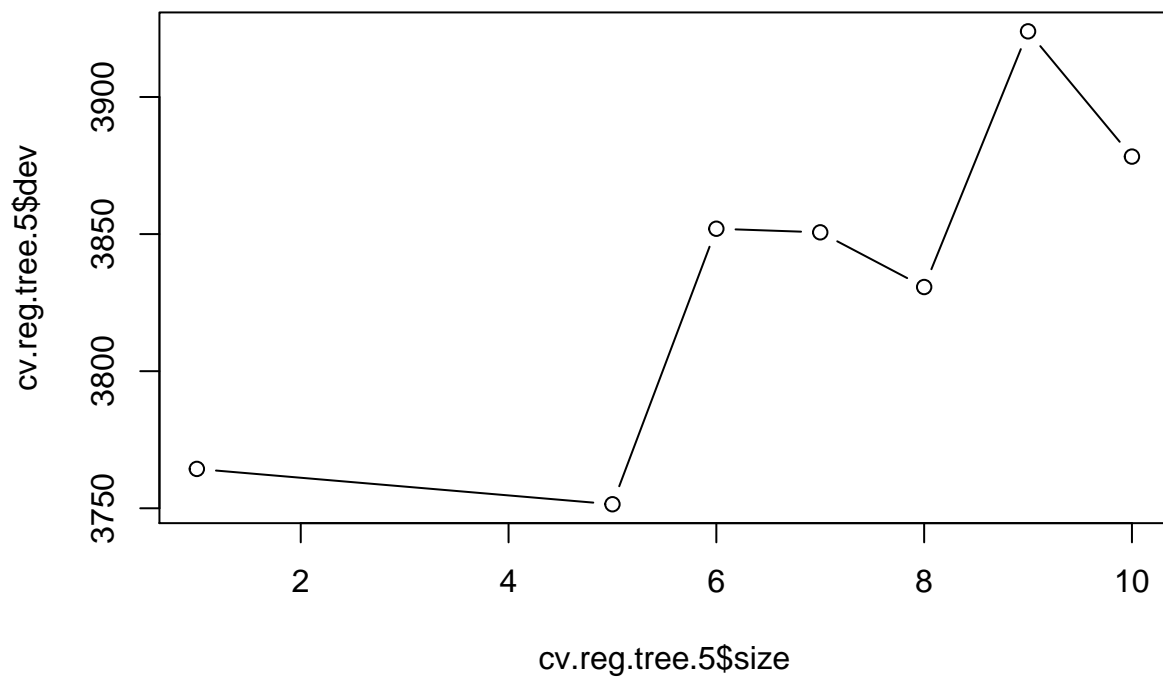
```

```
## Distribution of residuals:
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -10.6900 -1.5650 -0.2514  0.0000  1.8950  11.4900
```

```
plot(reg.tree.5)
text(reg.tree.5, pretty = 0)
```

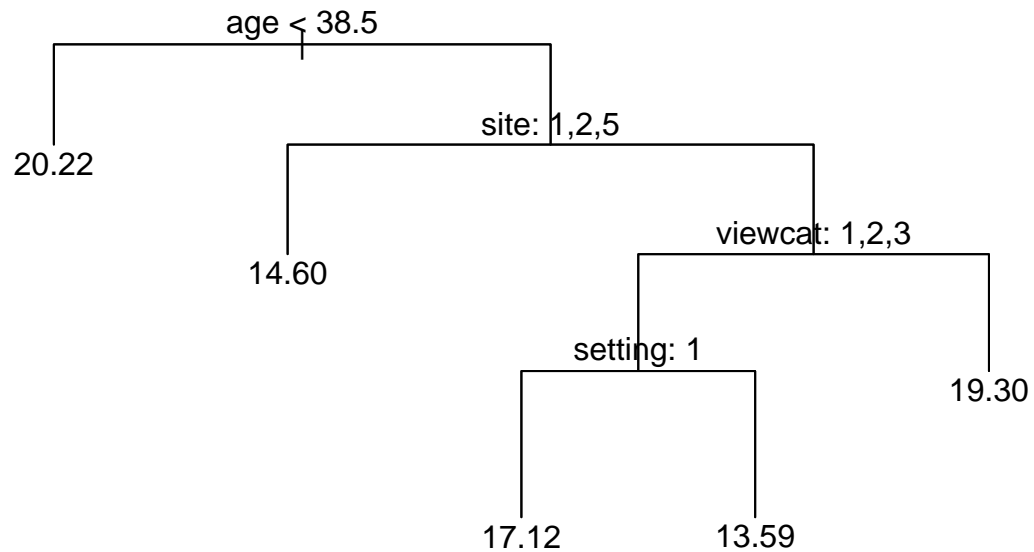


```
cv.reg.tree.5 <- cv.tree(reg.tree.5)
plot(cv.reg.tree.5$size, cv.reg.tree.5$dev, type = "b")
```



```
prune.reg.tree.5 <- prune.tree(reg.tree.5, best = 5)
plot(prune.reg.tree.5)
text(prune.reg.tree.5, pretty = 0)
```





```
yhat <- predict(prune.reg.tree.5, newdata = testing)
y.test <- testing[, "relatDiff"]

# Test MSE
mean((yhat-y.test)^2)
```

```
## [1] 19.88506
```

## Model 6

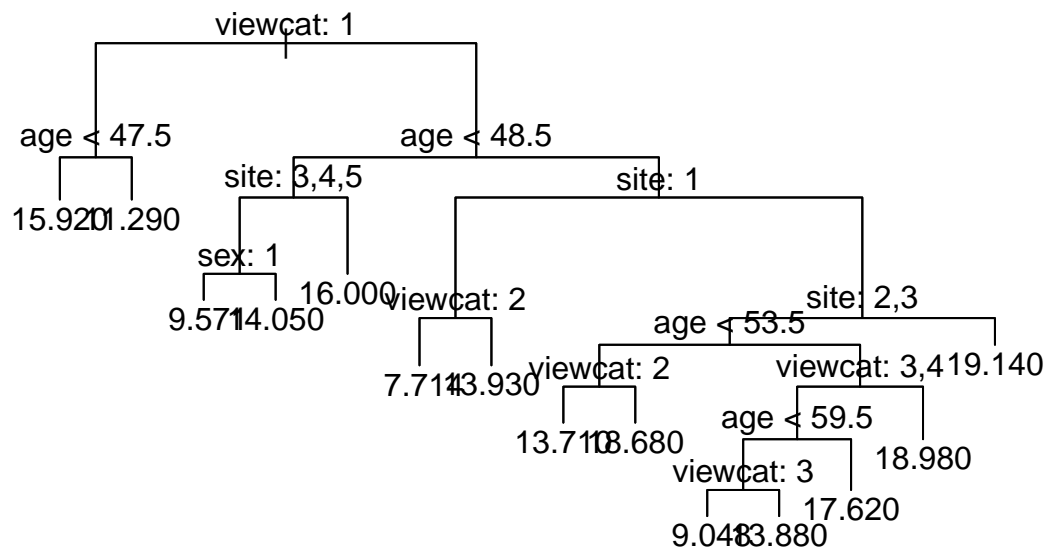
```
set.seed(1)

reg.tree.6 <- tree(clasfDiff ~ site + sex + age + viewcat + setting + viewenc, sesame.q1, subset = train)
summary(reg.tree.6)

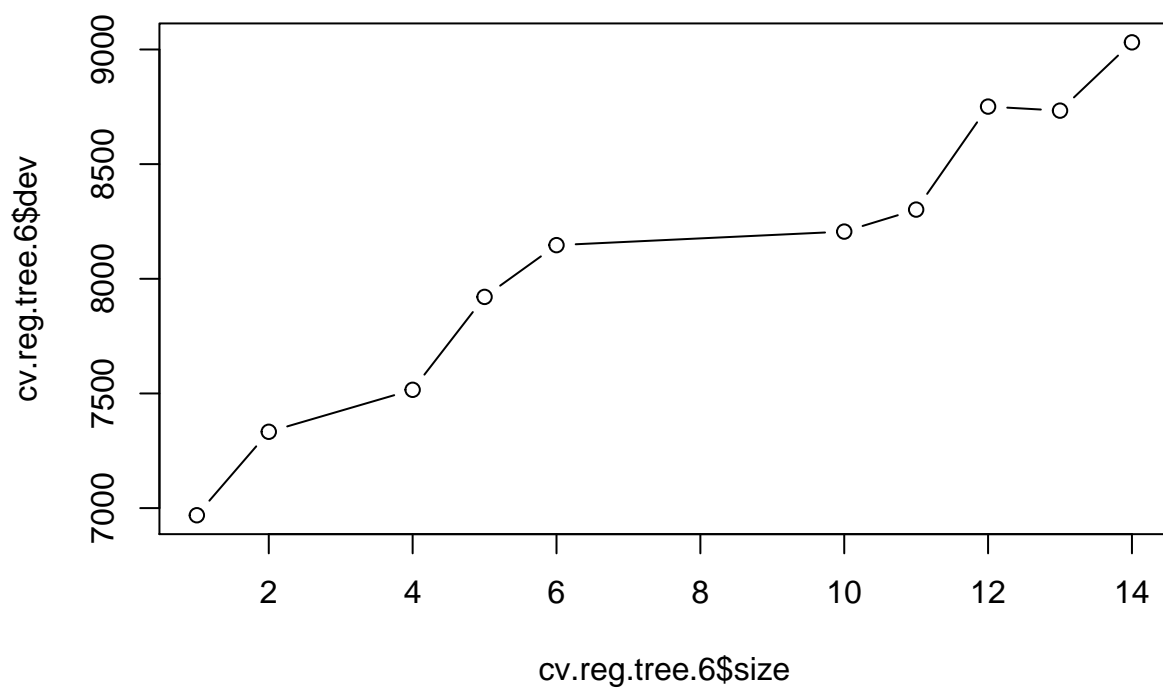
##
## Regression tree:
## tree(formula = clasfDiff ~ site + sex + age + viewcat + setting +
##       viewenc, data = sesame.q1, subset = train)
## Variables actually used in tree construction:
## [1] "viewcat" "age"      "site"      "sex"
## Number of terminal nodes:  14
## Residual mean deviance:  29.66 = 4568 / 154
```

```
## Distribution of residuals:
##      Min.   1st Qu.   Median     Mean   3rd Qu.     Max.
## -16.00000  -3.17300   0.01852   0.00000   3.71400   13.10000
```

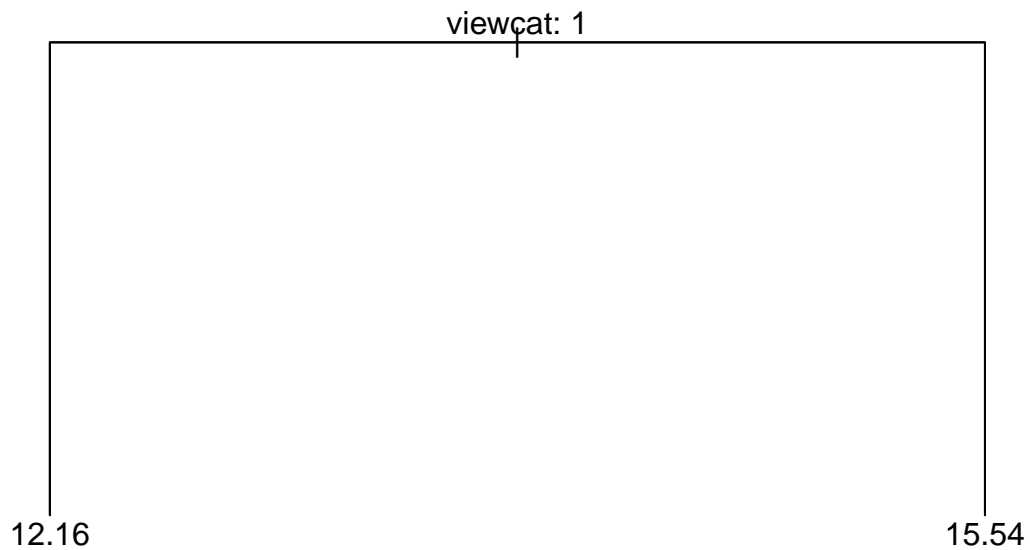
```
plot(reg.tree.6)
text(reg.tree.6, pretty = 0)
```



```
cv.reg.tree.6 <- cv.tree(reg.tree.6)
plot(cv.reg.tree.6$size, cv.reg.tree.6$dev, type = "b")
```



```
prune.reg.tree.6 <- prune.tree(reg.tree.6, best = 2)
plot(prune.reg.tree.6)
text(prune.reg.tree.6, pretty = 0)
```



```
yhat <- predict(prune.reg.tree.6, newdata = testing)
y.test <- testing[, "clasfDiff"]

# Test MSE
mean((yhat-y.test)^2)
```

```
## [1] 45.52784
```

```
df1 <- data.frame(Response = c("Changes in Body Parts Knolwedge Test Score", "Changes in Letters Test S
#
table <- kable(df1, caption = "Test Metrics", booktabs=T)
kable_styling(table, bootstrap_options = "striped", full_width = F, latex_options = "HOLD_position")
```

Table 1: Test Metrics

Response	Least.Reggression.Test.MSE	Ridge.Reggression.Test.MSE	Regression.Tr
Changes in Body Parts Knolwedge Test Score	32.45	21.61	
Changes in Letters Test Score	24.45	14.20	
Changes in Form Test Score	23.19	12.83	
Changes in Forms Test Knowledge	28.24	14.63	
Changes in Relational Terms Test Score	23.64	19.86	
Changes in Classification Skills Test Score	65.41	44.26	

**Q.2 Classification Question:** Can we use the pre-test scores and other demographic variables to predict which region the children came from?

SVM

```
set.seed(3241)

n <- nrow(sesame)
train.index <- sample(1:n, size = floor(0.7*n), replace=FALSE)
train.data <- sesame.sd[train.index,]
test.data <- sesame.sd[-train.index,]

train.data %>%
  count(site)
```

```
##   site  n
## 1     1 40
## 2     2 42
## 3     3 48
## 4     4 25
## 5     5 13
```

```
#1 60
#2 55
#3 64
#4 43
#5 18
```

```
total.weight <- 60+55+64+43+18
weight.1 <- total.weight/(5*60)
weight.2 <- total.weight/(5*55)
weight.3 <- total.weight/(5*64)
weight.4 <- total.weight/(5*43)
weight.5 <- total.weight/(5*18)
```

```
weight.4 <- 1.5
weight.5 <- 3
# Response: site (categorical)
set.seed(315)
costs <- c(0.001, 0.01, 0.1, 1, 5, 10, 100)
# c(0.1, 0.2, 0.5, 0.7, 1, 2, 3, 4)
gammas <- seq(0, 4, by=0.1)
```

```
linear.tune <- tune(svm, site~female+ male + sd_age+sd_pBod+sd_plet+sd_pform + sd_pnumb+sd_prelat+sd_pc,
  data=train.data, kernel="linear",
  ranges=list(cost=costs),
  class.weights=c("1"=weight.1,
                  "2"=weight.2,
                  "3"=weight.3,
                  "4"=weight.4,
                  "5"=weight.5),
```

```

class.type="one.versus.one")

radial.tune <- tune(svm, site~female + male + sd_age+sd_pBod+sd_plet+sd_pform + sd_pnumb+sd_prelat+sd_p
  data=train.data, kernel="radial",
  ranges=list(cost=costs,
              gamma=gammas),
  class.weights=c("1"=weight.1,
                  "2"=weight.2,
                  "3"=weight.3,
                  "4"=weight.4,
                  "5"=weight.5))
#radial.tune <- tune(svm, site~sex+age+prebody+prelet+preform+prenumb+prerelat+preclasf,
#  data=train.data, kernel="radial",
#  ranges=list(cost=costs,
#              gamma=gammas))

sigmoid.tune <- tune(svm, site~female + male + sd_age+sd_pBod+sd_plet+sd_pform + sd_pnumb+sd_prelat+sd_p
  data=train.data, kernel="sigmoid",
  ranges=list(cost=costs,
              gamma=gammas),
  class.weights=c("1"=weight.1,
                  "2"=weight.2,
                  "3"=weight.3,
                  "4"=weight.4,
                  "5"=weight.5))

```

```

linear.conMatrix <- table(true=test.data[, "site"],
  pred=predict(linear.tune$best.model, newdata=test.data))

radial.conMatrix <- table(true=test.data[, "site"],
  pred=predict(radial.tune$best.model, newdata=test.data))

sigmoid.conMatrix <- table(true=test.data[, "site"],
  pred=predict(sigmoid.tune$best.model, newdata=test.data))

confusionMatrix(linear.conMatrix)

```

```
## Confusion Matrix and Statistics
```

```
##
```

```
##      pred
```

```
## true  1  2  3  4  5
```

```
##      1  6  1  6  2  5
```

```
##      2  1  7  1  1  3
```

```
##      3  0  1 11  1  3
```

```
##      4  3  3  9  3  0
```

```
##      5  0  1  1  1  2
```

```
##
```

```
## Overall Statistics
```

```
##
```

```
##              Accuracy : 0.4028
```

```
##              95% CI : (0.2888, 0.525)
```

```
##      No Information Rate : 0.3889
```

```
##      P-Value [Acc > NIR] : 0.44844
```

```
##
##          Kappa : 0.2554
##
## McNemar's Test P-Value : 0.01728
##
## Statistics by Class:
##
##          Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity      0.60000  0.53846  0.3929  0.37500  0.15385
## Specificity      0.77419  0.89831  0.8864  0.76562  0.94915
## Pos Pred Value   0.30000  0.53846  0.6875  0.16667  0.40000
## Neg Pred Value   0.92308  0.89831  0.6964  0.90741  0.83582
## Prevalence       0.13889  0.18056  0.3889  0.11111  0.18056
## Detection Rate   0.08333  0.09722  0.1528  0.04167  0.02778
## Detection Prevalence 0.27778  0.18056  0.2222  0.25000  0.06944
## Balanced Accuracy 0.68710  0.71838  0.6396  0.57031  0.55150
```

```
confusionMatrix(radial.conMatrix)
```

```
## Confusion Matrix and Statistics
##
##      pred
## true  1  2  3  4  5
##      1  6  2  7  4  1
##      2  4  4  3  0  2
##      3  1  1 11  3  0
##      4  2  2  9  4  1
##      5  0  2  1  1  1
##
## Overall Statistics
##
##          Accuracy : 0.3611
##          95% CI : (0.2512, 0.4829)
##      No Information Rate : 0.4306
##      P-Value [Acc > NIR] : 0.9056
##
##          Kappa : 0.181
##
## McNemar's Test P-Value : 0.1807
##
## Statistics by Class:
##
##          Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity      0.46154  0.36364  0.3548  0.33333  0.20000
## Specificity      0.76271  0.85246  0.8780  0.76667  0.94030
## Pos Pred Value   0.30000  0.30769  0.6875  0.22222  0.20000
## Neg Pred Value   0.86538  0.88136  0.6429  0.85185  0.94030
## Prevalence       0.18056  0.15278  0.4306  0.16667  0.06944
## Detection Rate   0.08333  0.05556  0.1528  0.05556  0.01389
## Detection Prevalence 0.27778  0.18056  0.2222  0.25000  0.06944
## Balanced Accuracy 0.61213  0.60805  0.6164  0.55000  0.57015
```

```
confusionMatrix(sigmoid.conMatrix)
```

```
## Confusion Matrix and Statistics
##
##      pred
## true  1  2  3  4  5
##      1  1  1  6  9  3
##      2  1  3  4  5  0
##      3  1  0  4 11  0
##      4  0  0  7 11  0
##      5  0  0  3  2  0
##
## Overall Statistics
##
##              Accuracy : 0.2639
##              95% CI : (0.167, 0.381)
##      No Information Rate : 0.5278
##      P-Value [Acc > NIR] : 1
##
##              Kappa : 0.0434
##
##  McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##              Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity      0.33333  0.75000  0.16667  0.2895  0.00000
## Specificity      0.72464  0.85294  0.75000  0.7941  0.92754
## Pos Pred Value   0.05000  0.23077  0.25000  0.6111  0.00000
## Neg Pred Value   0.96154  0.98305  0.64286  0.5000  0.95522
## Prevalence       0.04167  0.05556  0.33333  0.5278  0.04167
## Detection Rate   0.01389  0.04167  0.05556  0.1528  0.00000
## Detection Prevalence 0.27778  0.18056  0.22222  0.2500  0.06944
## Balanced Accuracy 0.52899  0.80147  0.45833  0.5418  0.46377
```

```
predict(linear.tune$best.model, newdata=test.data)
```

```
##      6      7      8      9     12     16     19     22     24     25     29     38     42     43     46     48     54     56     57     59
##      1      4      1      1      5      1      1      2      5      5      5      3      1      4      3      3      3      5      3      3
##     64     82     84     88     92     93     94     98     99    100    104    106    113    116    118    120    125    127    128    137
##      5      1      4      2      2      2      2      2      5      3      5      2      2      4      3      3      3      5      3      3
##    139    143    145    152    156    162    164    172    176    180    181    182    183    187    188    189    190    191    192    193
##      3      2      3      3      5      3      3      3      5      3      3      4      1      2      3      3      4      3      3      2
##    196    202    207    208    212    221    222    225    226    227    234    235
##      1      2      3      3      4      3      1      5      3      4      5      2
## Levels: 1 2 3 4 5
```

```
predict(radial.tune$best.model, newdata=test.data)
```

```
##      6      7      8      9     12     16     19     22     24     25     29     38     42     43     46     48     54     56     57     59
##      3      3      1      1      2      4      4      4      3      3      3      1      2      1      5      3      4      1      1      3
```



```
## 64 82 84 88 92 93 94 98 99 100 104 106 113 116 118 120 125 127 128 137
## 3 1 1 2 1 2 5 2 5 3 3 2 1 1 4 3 3 3 4 3
## 139 143 145 152 156 162 164 172 176 180 181 182 183 187 188 189 190 191 192 193
## 3 4 3 3 3 3 3 3 2 3 3 3 3 2 3 3 3 3 4 2
## 196 202 207 208 212 221 222 225 226 227 234 235
## 4 1 1 4 5 3 4 2 3 4 5 2
## Levels: 1 2 3 4 5
```

```
predict(sigmoid.tune$best.model, newdata=test.data)
```

```
## 6 7 8 9 12 16 19 22 24 25 29 38 42 43 46 48 54 56 57 59
## 5 3 3 3 4 1 4 2 3 4 5 4 4 4 3 4 4 5 3 4
## 64 82 84 88 92 93 94 98 99 100 104 106 113 116 118 120 125 127 128 137
## 4 3 4 3 3 2 1 2 4 4 4 3 2 4 4 3 4 4 3 4
## 139 143 145 152 156 162 164 172 176 180 181 182 183 187 188 189 190 191 192 193
## 4 3 4 4 1 4 3 4 4 4 4 4 4 3 4 4 4 3 3 4
## 196 202 207 208 212 221 222 225 226 227 234 235
## 3 4 4 3 3 4 3 3 4 3 4 3
## Levels: 1 2 3 4 5
```

```
test.data$site
```

```
## [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3
## [39] 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5
## Levels: 1 2 3 4 5
```

Radial kernel improves prediction on class 1.

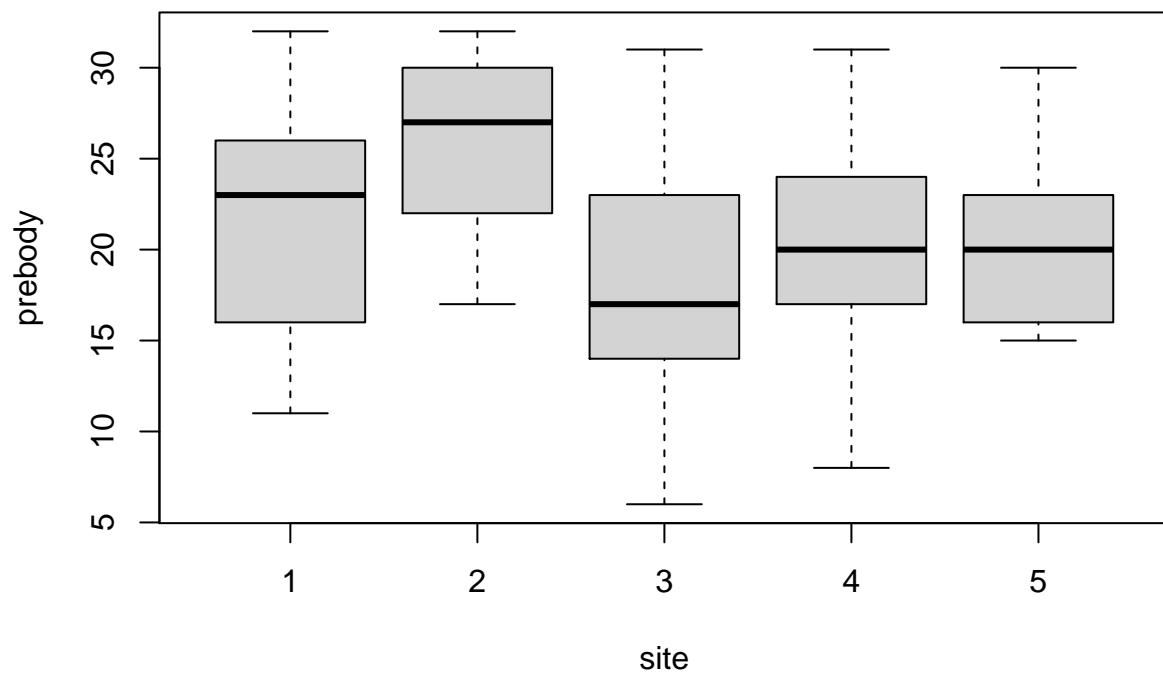
RBF slightly improved after standardizing? (it seems slightly more likely to predict on class 1. ) thought, simpler models still retain the same performance (arguably better) `sd_age+sd_pBod+sd_plet`. But we are still not getting any prediction on class 4 & 5.

After assign class weights using this formula:

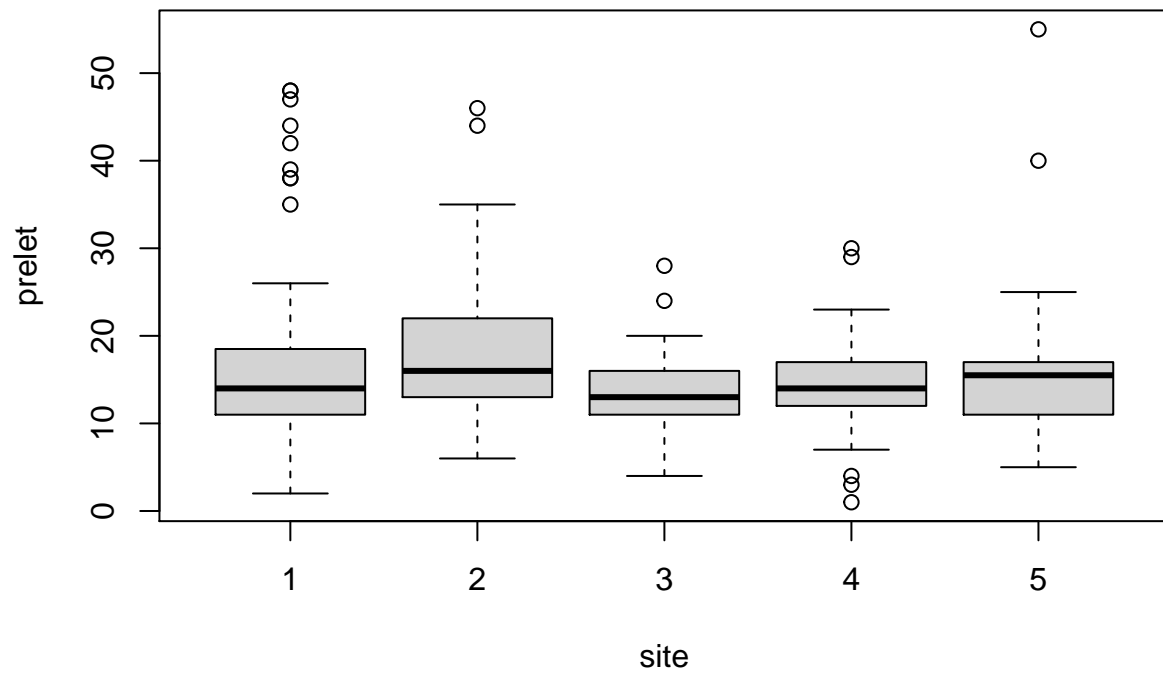
$$w_j = \frac{n}{kn_j}, \text{ n is total number of data points, k is number of classes}$$

Our model begins to make predictions on class 4 & class 5, though at the cost of overall accuracy. If we increase the weight for 4 & 5 to 1.5 and 3 respectively the performance of Radial SVM decreases but that of linear SVM increases to be comparable to Radial SVM's recorded highest accuracy (a little bit over 0.40).

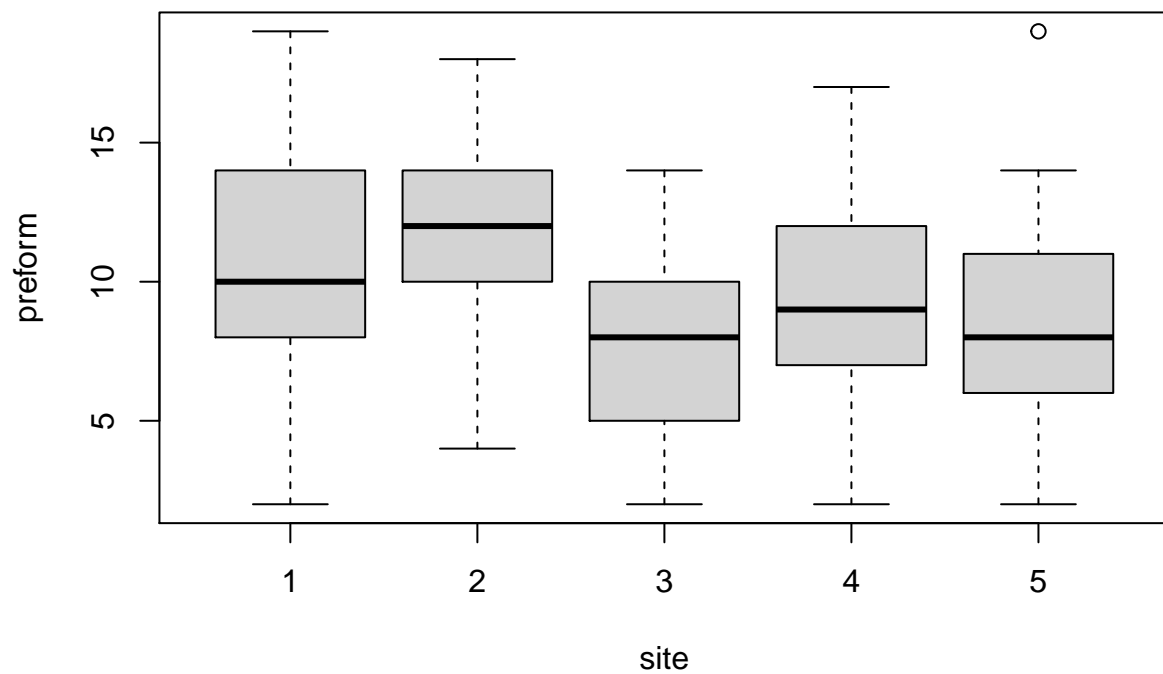
```
# trying to do more EDA to see if anything explains why the data is not linearly separable
boxplot(prebody~site, data=sesame)
```



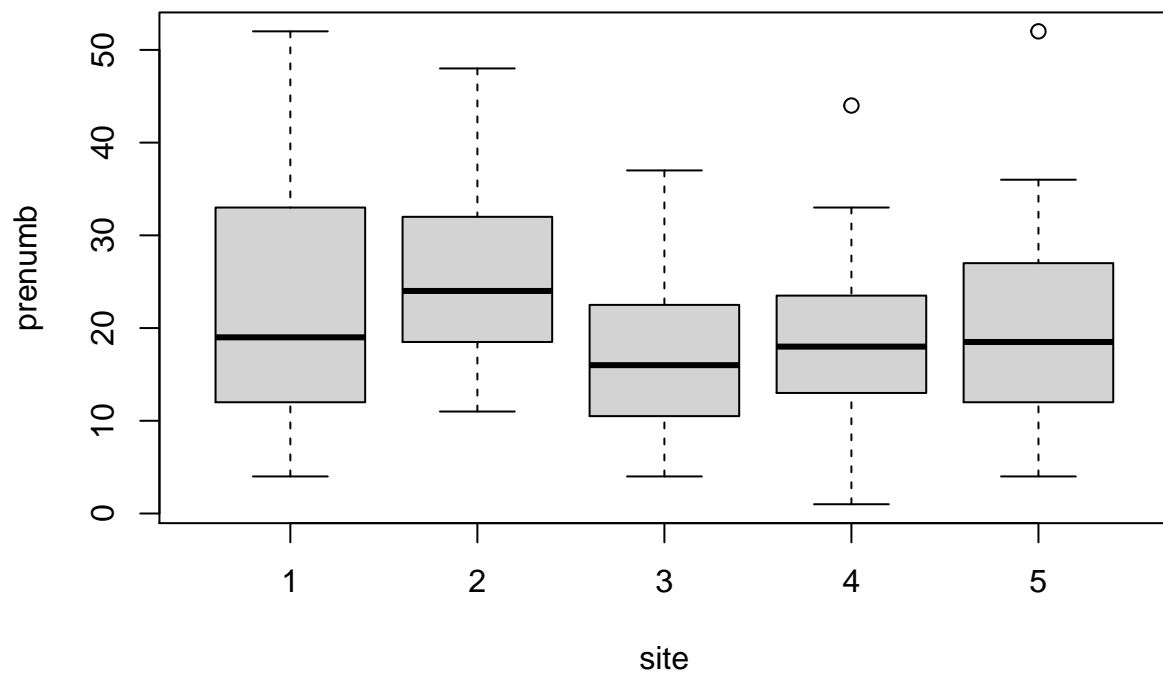
```
boxplot(prelet~site, data=sesame)
```



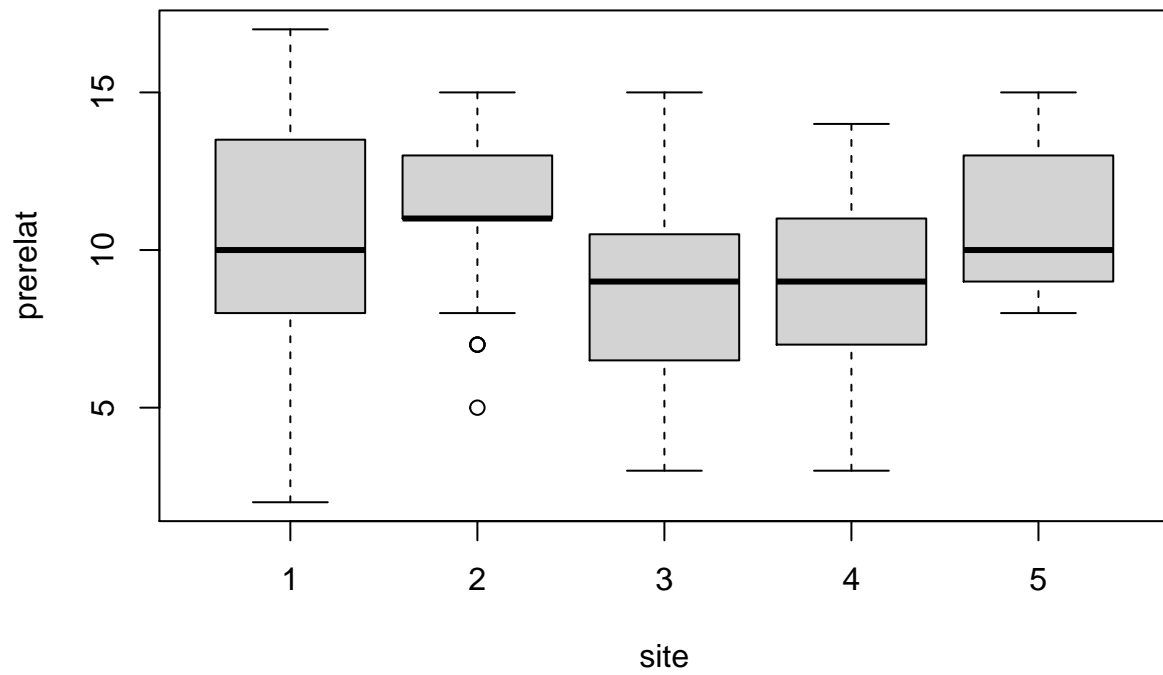
```
boxplot(preform~site, data=sesame)
```



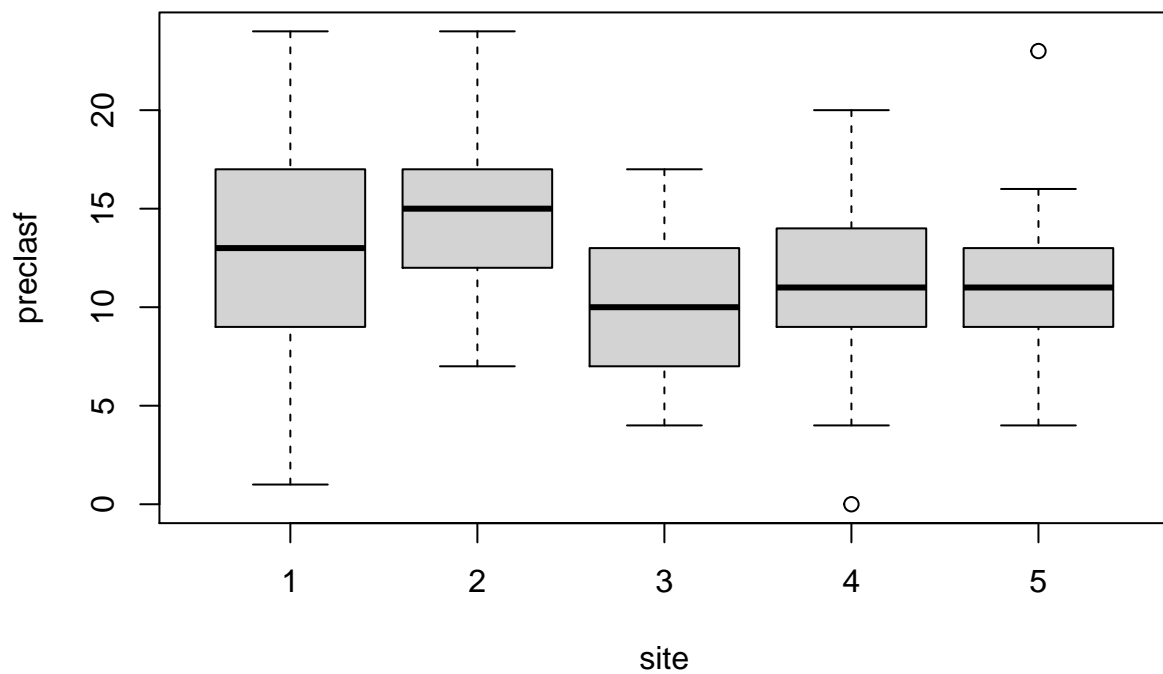
```
boxplot(preform~site, data=sesame)
```



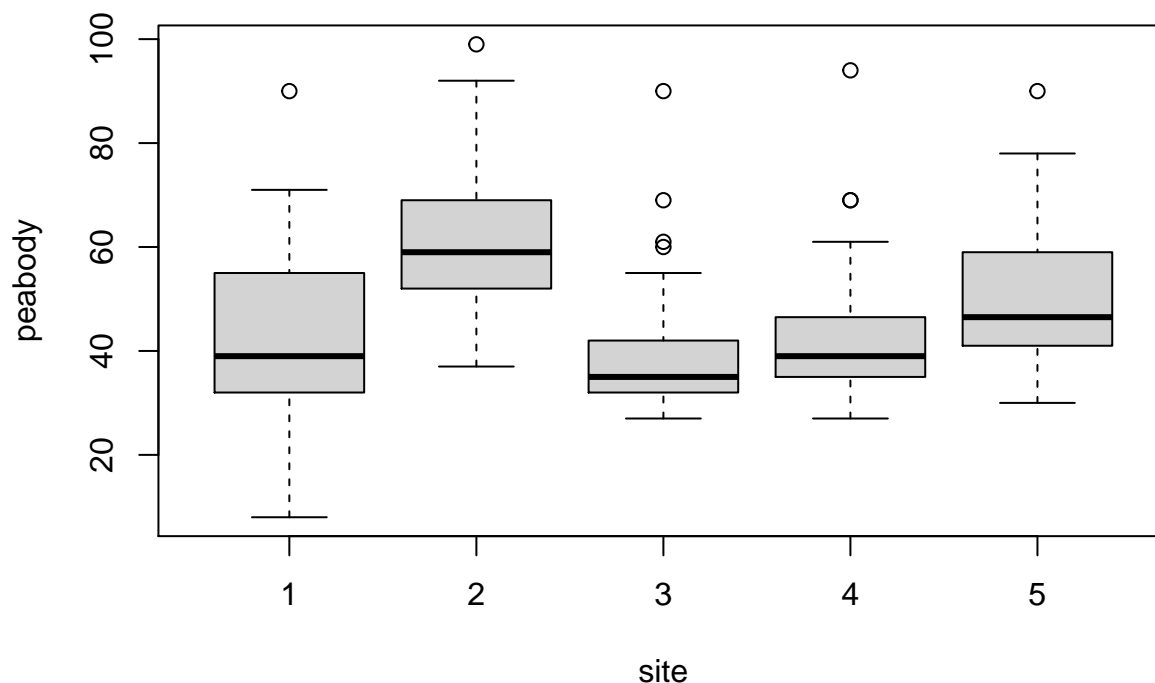
```
boxplot(prerelat~site, data=sesame)
```



```
boxplot(preclasf~site, data=sesame)
```



```
boxplot(peabody~site, data=sesame)
```



## Trees

```
set.seed(3215)
#tree.data <- sesame %>%
# select(site, sex, age, viewcat, setting, viewenc, prebody, prelet, preform,
#        prenumb, prerelat, preclasf)

n <- nrow(sesame)
train.index <- sample(1:n, size = floor(0.7*n), replace=FALSE)
#train.tree <- tree.data[train.index,]
#test.tree <- tree.data[-train.index,]
# "viewcat", "setting", "viewenc",

# ,"prebody", "prelet","preform", "prenumb", "prerelat", "preclasf", "postbody", "postlet", "postform",

tree.features <- c("site", "age", "viewcat", "setting", "viewenc")

tree.data <- sesame[, tree.features]
train.data <- tree.data[train.index,]
test.data <- tree.data[-train.index,]

rf.tree<- randomForest(site~., data=tree.data, subset=train.index,
                        mtry=4, importance=TRUE)
```



```
importance(rf.tree)
```

```
##           1           2           3           4           5 MeanDecreaseAccuracy
## age      16.071180 11.5639655  1.533526 10.24956 -12.646482      15.477245
## viewcat   1.412544  8.8509648  4.714890 26.85440  -5.674656      19.084980
## setting   9.950184  0.7116516 11.570042 20.04012   6.879913      22.690481
## viewenc   1.061428 -1.4942151  6.283556 -2.36010  -3.154027       1.353554
##      MeanDecreaseGini
## age              67.80480
## viewcat          24.57478
## setting          10.77912
## viewenc          10.03605
```

```
rf.pred <- predict(rf.tree, newdata=test.data)
```

```
tree.conMatrix <- table(true=test.data[, "site"],
                        pred=rf.pred)
confusionMatrix(tree.conMatrix)
```

```
## Confusion Matrix and Statistics
```

```
##
##      pred
## true  1  2  3  4  5
##   1 10  2  3  3  0
##   2  4 10  1  1  2
##   3  2  3 11  3  0
##   4  1  1  3  5  2
##   5  0  1  3  0  1
```

```
## Overall Statistics
```

```
##
##              Accuracy : 0.5139
##              95% CI : (0.3931, 0.6335)
##      No Information Rate : 0.2917
##      P-Value [Acc > NIR] : 6.203e-05
```

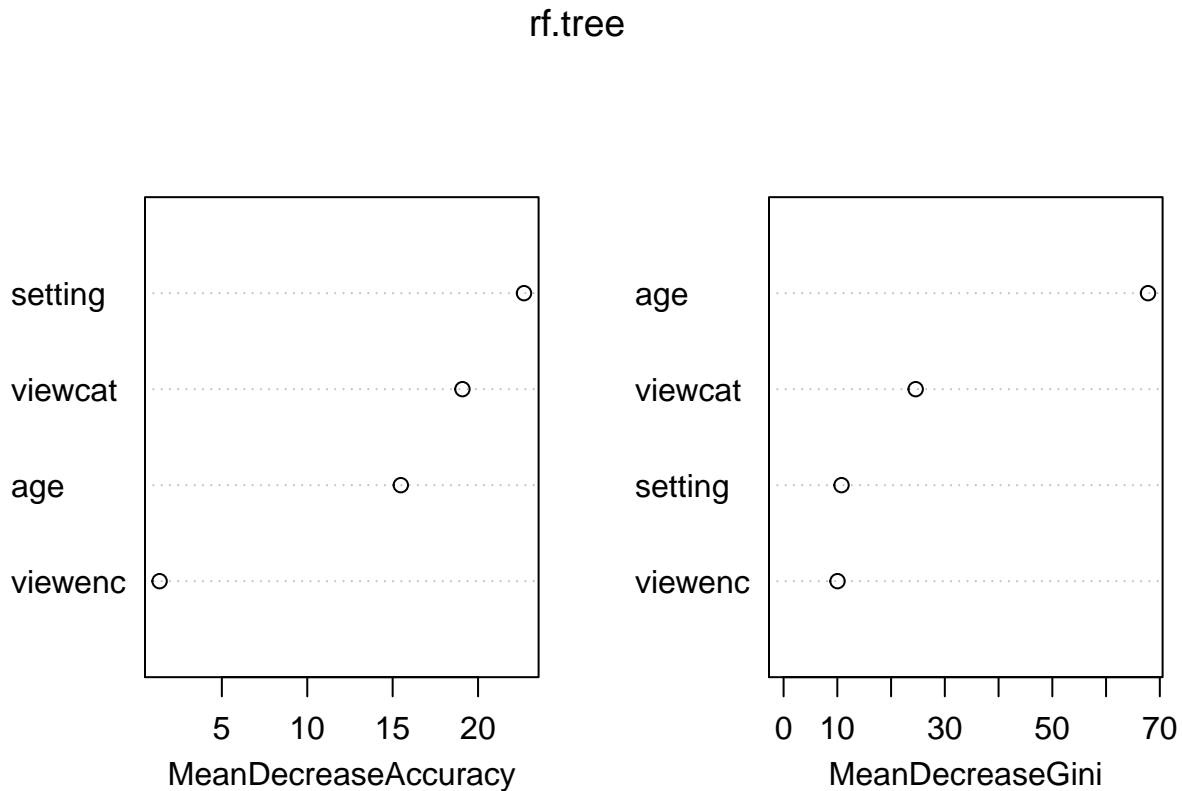
```
##
##              Kappa : 0.3706
```

```
##
## McNemar's Test P-Value : NA
```

```
## Statistics by Class:
```

```
##
##              Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity      0.5882   0.5882   0.5238  0.41667  0.20000
## Specificity      0.8545   0.8545   0.8431  0.88333  0.94030
## Pos Pred Value    0.5556   0.5556   0.5789  0.41667  0.20000
## Neg Pred Value    0.8704   0.8704   0.8113  0.88333  0.94030
## Prevalence       0.2361   0.2361   0.2917  0.16667  0.06944
## Detection Rate    0.1389   0.1389   0.1528  0.06944  0.01389
## Detection Prevalence 0.2500   0.2500   0.2639  0.16667  0.06944
## Balanced Accuracy 0.7214   0.7214   0.6835  0.65000  0.57015
```

```
varImpPlot(rf.tree)
```



0.42 – 0.5139 (but not including the test scores.) around 0.45-0.48, when including the pretest scores.

As seen in the table above, there is a notable discrepancy in the number of observations that lay in classes 4 and 5 for the variable `site`. More specifically, in the training data, there are just 25 observations with a value of 4 for the variable `site` and just 13 observations with a value of 5 for the variable `site`. In other words, there are less disadvantaged rural children and disadvantaged Spanish speaking children.

Consequently, when we initially ran our random forest model, our model was performing worse for test observations that take on the values 4 or 5 for the variable `site`.

To remedy this problem, we decided to use Synthetic Minority Oversampling Technique (SMOTE). SMOTE works by generating new samples in the classes of the response variable that are less represented. These new samples are generated using linear combinations of the “k” nearest neighbors in a given class. In this instance, we set  $k = 5$ .

```
train.data$age <- as.numeric(train.data$age)
train.data$viewcat <- as.numeric(train.data$viewcat)
train.data$setting <- as.numeric(train.data$setting)
train.data$viewenc <- as.factor(train.data$viewenc)

test.data$age <- as.numeric(test.data$age)
test.data$viewcat <- as.numeric(test.data$viewcat)
test.data$setting <- as.numeric(test.data$setting)
test.data$viewenc <- as.factor(test.data$viewenc)
```

```
balanced.train.data <- SmoteClassif(site ~ ., train.data, k = 5, repl = FALSE, dist = "HEOM")
# k --> represents the number of nearest neighbors (5) used to generate new examples of the minority
# repl = FALSE --> cannot have repetition of examples when performing under-sampling by selecting amo
```

```
balanced.train.data %>%
  count(site)
```

```
##   site  n
## 1    1 34
## 2    2 34
## 3    3 34
## 4    4 33
## 5    5 34
```

```
rf.tree<- randomForest(site~., data=balanced.train.data,
                        mtry=4, importance=TRUE)
```

```
importance(rf.tree)
```

```
##           1           2           3           4           5 MeanDecreaseAccuracy
## age      15.112165 11.5044815 2.831522 12.2530454 12.35896      24.534350
## viewcat   2.482319 12.5671060 5.114628 28.5410065 19.94849      32.088862
## setting  14.672536  0.4233347 5.467389 22.8438707 32.97928      34.377861
## viewenc   5.604315 -2.5081007 6.322412  0.1078161 12.91417      9.960821
##           MeanDecreaseGini
## age              73.124641
## viewcat          29.595993
## setting           9.359814
## viewenc           9.254742
```

```
rf.pred <- predict(rf.tree, newdata=test.data)
```

```
tree.conMatrix <- table(true=test.data[, "site"],
                        pred=rf.pred)
confusionMatrix(tree.conMatrix)
```

```
## Confusion Matrix and Statistics
```

```
##
##      pred
## true 1 2 3 4 5
##    1 7 4 3 2 2
##    2 4 8 1 2 3
##    3 3 4 9 2 1
##    4 0 2 3 5 2
##    5 0 2 2 0 1
##
```

```
## Overall Statistics
```

```
##
##           Accuracy : 0.4167
##           95% CI : (0.3015, 0.5389)
##    No Information Rate : 0.2778
##    P-Value [Acc > NIR] : 0.007741
```

```
##
##           Kappa : 0.2539
##
## Mcnemar's Test P-Value : 0.576888
##
## Statistics by Class:
##
##           Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity      0.50000   0.4000   0.5000  0.45455  0.11111
## Specificity      0.81034   0.8077   0.8148  0.88525  0.93651
## Pos Pred Value   0.38889   0.4444   0.4737  0.41667  0.20000
## Neg Pred Value   0.87037   0.7778   0.8302  0.90000  0.88060
## Prevalence       0.19444   0.2778   0.2500  0.15278  0.12500
## Detection Rate   0.09722   0.1111   0.1250  0.06944  0.01389
## Detection Prevalence 0.25000   0.2500   0.2639  0.16667  0.06944
## Balanced Accuracy 0.65517   0.6038   0.6574  0.66990  0.52381
```

While the `SmoteClassif()` function certainly did its job by balancing out the number of observations for each value of `site` in the training data, the new random forest model (fitted to this new data set) is less accurate and sees little improvement in the detection of `site` values of 4 and 5.

```
# set.seed(3215)
#
# # , "prebody", "prelet", "preform", "prenumb", "prerelat", "preclasf", "postbody", "postlet", "postform",
#
#
#
# features <- c("site", "age", "viewcat", "setting", "viewenc", "prebody", "prelet", "preform", "prenumb",
#
# tree.2 <- sesame[, features]
# train.2 <- tree.2[train.index,]
# test.2 <- tree.2[-train.index,]
#
# boost.tree <- gbm(site ~., data=train.2,
#                   distribution="multinomial", n.trees=5000,
#                   interaction.depth=1)
#
# # y.boost <- table(true=test.2[, "site"],
# #                 pred=predict(boost.tree, newdata=test.2))
#
# boost.conMatrix <- table(true=test.2$site,
#                          pred=predict(boost.tree, newdata=test.2))
# confusionMatrix(boost.conMatrix)
```

## Logistic Regression

```
# , "prebody", "prelet", "preform", "prenumb", "prerelat", "preclasf", "postbody", "postlet", "postform",
tree.features <- c("site", "age", "viewcat", "setting", "viewenc", "prebody", "prelet", "preform", "prenumb",
tree.log <- sesame[, tree.features]
train.log <- tree.log[train.index,]
```

```
test.log <- tree.log[-train.index,]
```

```
multinom.log <- multinom(factor(site)~., data=train.log)
```

```
## # weights: 60 (44 variable)
## initial value 270.385569
## iter 10 value 224.661284
## iter 20 value 205.001057
## iter 30 value 193.994082
## iter 40 value 186.948903
## iter 50 value 186.213534
## iter 60 value 186.148958
## iter 70 value 186.146812
## iter 70 value 186.146811
## iter 70 value 186.146811
## final value 186.146811
## converged
```

```
summary(multinom.log)
```

```
## Call:
## multinom(formula = factor(site) ~ ., data = train.log)
##
## Coefficients:
## (Intercept)      age      viewcat      setting      viewenc      prebody
## 2 -2.33155723 -0.02851059  0.1671689  -0.1723663 -0.4205104  0.18169812
## 3  0.05755516  0.20107446  0.0726002  -1.4396594 -1.5937567 -0.10132305
## 4  0.52070488  0.14041265 -1.1560807  -0.2744569 -1.3458286 -0.08902983
## 5 10.77382380  0.20246633 -0.1290428 -16.3009311 -1.5011369 -0.09094212
##      prelet      preform      prenumb      prerelat      preclasf
## 2 -0.021574850 -0.12985947 -0.0781747329  0.16740257  0.10875790
## 3 -0.009618286 -0.32080214 -0.0307718697  0.05942666 -0.08062801
## 4 -0.023880439  0.05801188 -0.0008763081 -0.11001706 -0.01601207
## 5  0.149877507 -0.31580541 -0.1020246788  0.34854595 -0.12639855
##
## Std. Errors:
## (Intercept)      age      viewcat      setting      viewenc      prebody      prelet
## 2  2.539808 0.05877122 0.2735599 0.6038785 0.6013929 0.07166225 0.03755288
## 3  2.552263 0.06296057 0.2800905 0.6392345 0.6272709 0.07157933 0.05105309
## 4  2.743404 0.06311748 0.3228949 0.6489907 0.6712331 0.07501958 0.04404500
## 5  2.206224 0.08353394 0.4579698 2.2063318 0.9368912 0.09992158 0.07033801
##      preform      prenumb      prerelat      preclasf
## 2 0.1144522 0.05108224 0.1361291 0.08630272
## 3 0.1256428 0.05602075 0.1440701 0.09338212
## 4 0.1173569 0.05356710 0.1389164 0.09183664
## 5 0.1771026 0.08446858 0.2087433 0.14707062
##
## Residual Deviance: 372.2936
## AIC: 460.2936
```

```
tabs <- table(true=test.log[, "site"],
              pred=predict(multinom.log,newdata=test.log))
confusionMatrix(tabs)
```

```

## Confusion Matrix and Statistics
##
##      pred
## true  1  2  3  4  5
##      1  3  4  8  3  0
##      2  8  7  2  1  0
##      3  2  3 12  2  0
##      4  0  1  4  6  1
##      5  0  0  2  1  2
##
## Overall Statistics
##
##              Accuracy : 0.4167
##              95% CI : (0.3015, 0.5389)
##      No Information Rate : 0.3889
##      P-Value [Acc > NIR] : 0.3557
##
##              Kappa : 0.2396
##
##  McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##              Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity      0.23077  0.46667  0.4286  0.46154  0.66667
## Specificity      0.74576  0.80702  0.8409  0.89831  0.95652
## Pos Pred Value   0.16667  0.38889  0.6316  0.50000  0.40000
## Neg Pred Value   0.81481  0.85185  0.6981  0.88333  0.98507
## Prevalence       0.18056  0.20833  0.3889  0.18056  0.04167
## Detection Rate   0.04167  0.09722  0.1667  0.08333  0.02778
## Detection Prevalence 0.25000  0.25000  0.2639  0.16667  0.06944
## Balanced Accuracy 0.48827  0.63684  0.6347  0.67992  0.81159

##forward selection
log.tune <- train(site~(.)^2, data=train.log, method="multinom", direction="forward",
                  k=log(3562))

## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 223.009710
## iter  20 value 185.817438
## iter  30 value 164.746276
## iter  40 value 142.208692
## iter  50 value 114.120976
## iter  60 value 105.061057
## iter  70 value 96.136366
## iter  80 value 86.279565
## iter  90 value 68.226344
## iter 100 value 56.986576
## final  value 56.986576
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 223.009765

```

```

## iter 20 value 185.818463
## iter 30 value 164.749701
## iter 40 value 142.227179
## iter 50 value 114.217355
## iter 60 value 105.233432
## iter 70 value 96.390732
## iter 80 value 85.327276
## iter 90 value 70.002063
## iter 100 value 59.270526
## final value 59.270526
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 223.009710
## iter 20 value 185.817439
## iter 30 value 164.746280
## iter 40 value 142.208711
## iter 50 value 114.121072
## iter 60 value 105.061229
## iter 70 value 96.136613
## iter 80 value 86.280792
## iter 90 value 68.229713
## iter 100 value 56.993284
## final value 56.993284
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 173.938475
## iter 20 value 138.721492
## iter 30 value 121.837401
## iter 40 value 102.694977
## iter 50 value 87.793408
## iter 60 value 78.834826
## iter 70 value 69.705169
## iter 80 value 60.078338
## iter 90 value 49.433061
## iter 100 value 30.412977
## final value 30.412977
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 173.938525
## iter 20 value 138.722238
## iter 30 value 121.840535
## iter 40 value 102.713762
## iter 50 value 87.856776
## iter 60 value 78.916581
## iter 70 value 71.245467
## iter 80 value 61.578500
## iter 90 value 49.020813
## iter 100 value 35.855955
## final value 35.855955
## stopped after 100 iterations
## # weights: 285 (224 variable)

```

```

## initial value 270.385569
## iter 10 value 173.938475
## iter 20 value 138.721493
## iter 30 value 121.837404
## iter 40 value 102.694996
## iter 50 value 87.793471
## iter 60 value 78.834899
## iter 70 value 69.705169
## iter 80 value 60.081440
## iter 90 value 49.429589
## iter 100 value 30.613096
## final value 30.613096
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 196.128288
## iter 20 value 152.338821
## iter 30 value 134.098144
## iter 40 value 120.528256
## iter 50 value 107.006267
## iter 60 value 97.433053
## iter 70 value 91.319595
## iter 80 value 80.124846
## iter 90 value 66.699682
## iter 100 value 51.368025
## final value 51.368025
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 196.128329
## iter 20 value 152.339603
## iter 30 value 134.101185
## iter 40 value 120.542755
## iter 50 value 107.075877
## iter 60 value 97.622659
## iter 70 value 90.390179
## iter 80 value 79.683704
## iter 90 value 68.002823
## iter 100 value 53.884570
## final value 53.884570
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 196.128288
## iter 20 value 152.338822
## iter 30 value 134.098147
## iter 40 value 120.528271
## iter 50 value 107.006336
## iter 60 value 97.433244
## iter 70 value 91.320015
## iter 80 value 80.124455
## iter 90 value 66.695323
## iter 100 value 52.559314
## final value 52.559314

```



```

## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 189.587833
## iter 20 value 163.400377
## iter 30 value 149.397008
## iter 40 value 130.865745
## iter 50 value 118.037225
## iter 60 value 102.868952
## iter 70 value 92.450294
## iter 80 value 81.333564
## iter 90 value 66.847399
## iter 100 value 54.770259
## final value 54.770259
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 189.587869
## iter 20 value 163.400999
## iter 30 value 149.399651
## iter 40 value 130.879708
## iter 50 value 118.074280
## iter 60 value 103.060569
## iter 70 value 92.686823
## iter 80 value 82.398605
## iter 90 value 70.243125
## iter 100 value 56.884732
## final value 56.884732
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 189.587833
## iter 20 value 163.400378
## iter 30 value 149.397010
## iter 40 value 130.865759
## iter 50 value 118.037262
## iter 60 value 102.869143
## iter 70 value 92.450438
## iter 80 value 81.335289
## iter 90 value 66.849983
## iter 100 value 54.773938
## final value 54.773938
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 218.066994
## iter 20 value 166.160778
## iter 30 value 147.813821
## iter 40 value 126.512571
## iter 50 value 105.678336
## iter 60 value 92.005056
## iter 70 value 76.732392
## iter 80 value 65.655736
## iter 90 value 52.831643

```

```

## iter 100 value 41.316980
## final value 41.316980
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 218.067049
## iter 20 value 166.161877
## iter 30 value 147.817302
## iter 40 value 126.532313
## iter 50 value 106.035090
## iter 60 value 92.573822
## iter 70 value 77.440839
## iter 80 value 66.671660
## iter 90 value 57.517903
## iter 100 value 46.862519
## final value 46.862519
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 218.066994
## iter 20 value 166.160780
## iter 30 value 147.813825
## iter 40 value 126.512591
## iter 50 value 105.678677
## iter 60 value 92.005568
## iter 70 value 76.732185
## iter 80 value 65.656811
## iter 90 value 52.829713
## iter 100 value 41.436202
## final value 41.436202
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 187.347617
## iter 20 value 167.486745
## iter 30 value 154.146359
## iter 40 value 139.480057
## iter 50 value 122.619656
## iter 60 value 110.529375
## iter 70 value 102.625116
## iter 80 value 90.421159
## iter 90 value 74.054788
## iter 100 value 57.909809
## final value 57.909809
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 187.347643
## iter 20 value 167.487082
## iter 30 value 154.148036
## iter 40 value 139.492855
## iter 50 value 122.826896
## iter 60 value 110.867861
## iter 70 value 102.854835

```

```

## iter 80 value 92.576101
## iter 90 value 77.396820
## iter 100 value 64.941943
## final value 64.941943
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 187.347617
## iter 20 value 167.486745
## iter 30 value 154.146360
## iter 40 value 139.480070
## iter 50 value 122.619857
## iter 60 value 110.529704
## iter 70 value 102.625311
## iter 80 value 90.422822
## iter 90 value 74.054983
## iter 100 value 57.934478
## final value 57.934478
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 208.590047
## iter 20 value 174.245490
## iter 30 value 157.555386
## iter 40 value 129.021489
## iter 50 value 109.837072
## iter 60 value 93.512606
## iter 70 value 81.991673
## iter 80 value 74.052179
## iter 90 value 60.298671
## iter 100 value 48.252392
## final value 48.252392
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 208.590094
## iter 20 value 174.246094
## iter 30 value 157.557920
## iter 40 value 129.041017
## iter 50 value 110.099794
## iter 60 value 95.257380
## iter 70 value 84.712232
## iter 80 value 76.674761
## iter 90 value 63.088355
## iter 100 value 48.790905
## final value 48.790905
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 208.590047
## iter 20 value 174.245490
## iter 30 value 157.555389
## iter 40 value 129.021509
## iter 50 value 109.837330

```

```

## iter 60 value 93.513161
## iter 70 value 81.992850
## iter 80 value 74.053951
## iter 90 value 60.302615
## iter 100 value 48.251118
## final value 48.251118
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 198.318184
## iter 20 value 175.950870
## iter 30 value 165.501135
## iter 40 value 147.679228
## iter 50 value 135.519155
## iter 60 value 124.002190
## iter 70 value 113.636452
## iter 80 value 99.187196
## iter 90 value 80.973452
## iter 100 value 67.940817
## final value 67.940817
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 198.318211
## iter 20 value 175.951353
## iter 30 value 165.503279
## iter 40 value 147.694380
## iter 50 value 135.553180
## iter 60 value 124.189654
## iter 70 value 114.073441
## iter 80 value 100.694573
## iter 90 value 84.427530
## iter 100 value 69.210273
## final value 69.210273
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 198.318184
## iter 20 value 175.950870
## iter 30 value 165.501137
## iter 40 value 147.679243
## iter 50 value 135.519189
## iter 60 value 124.002378
## iter 70 value 113.636895
## iter 80 value 99.188748
## iter 90 value 80.977323
## iter 100 value 67.968766
## final value 67.968766
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 195.878402
## iter 20 value 160.411948
## iter 30 value 146.293438

```

```

## iter 40 value 132.445684
## iter 50 value 121.735515
## iter 60 value 110.913748
## iter 70 value 101.797034
## iter 80 value 89.943451
## iter 90 value 76.345444
## iter 100 value 66.319285
## final value 66.319285
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 195.878435
## iter 20 value 160.412633
## iter 30 value 146.295768
## iter 40 value 132.456699
## iter 50 value 121.753991
## iter 60 value 111.068167
## iter 70 value 101.725838
## iter 80 value 91.901527
## iter 90 value 79.790681
## iter 100 value 67.956986
## final value 67.956986
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 195.878402
## iter 20 value 160.411948
## iter 30 value 146.293441
## iter 40 value 132.445695
## iter 50 value 121.735534
## iter 60 value 110.913903
## iter 70 value 101.797552
## iter 80 value 89.944796
## iter 90 value 76.348390
## iter 100 value 66.328651
## final value 66.328651
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 195.507155
## iter 20 value 168.008151
## iter 30 value 154.351529
## iter 40 value 139.161429
## iter 50 value 126.127147
## iter 60 value 115.398471
## iter 70 value 107.554452
## iter 80 value 97.821720
## iter 90 value 79.668626
## iter 100 value 66.257381
## final value 66.257381
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 195.507174

```

```

## iter 20 value 168.008683
## iter 30 value 154.353641
## iter 40 value 139.169359
## iter 50 value 126.120865
## iter 60 value 115.561001
## iter 70 value 107.881745
## iter 80 value 98.577837
## iter 90 value 81.756531
## iter 100 value 68.196678
## final value 68.196678
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 195.507155
## iter 20 value 168.008152
## iter 30 value 154.351531
## iter 40 value 139.161437
## iter 50 value 126.127140
## iter 60 value 115.398638
## iter 70 value 107.554789
## iter 80 value 97.822265
## iter 90 value 79.670619
## iter 100 value 66.258275
## final value 66.258275
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 216.166446
## iter 20 value 185.522611
## iter 30 value 173.564102
## iter 40 value 155.099978
## iter 50 value 139.144040
## iter 60 value 129.027054
## iter 70 value 119.722624
## iter 80 value 107.450564
## iter 90 value 91.208985
## iter 100 value 78.111012
## final value 78.111012
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 216.166467
## iter 20 value 185.523132
## iter 30 value 173.566575
## iter 40 value 155.113493
## iter 50 value 139.192277
## iter 60 value 128.900937
## iter 70 value 119.118226
## iter 80 value 107.573476
## iter 90 value 94.029691
## iter 100 value 82.140576
## final value 82.140576
## stopped after 100 iterations
## # weights: 285 (224 variable)

```

```

## initial value 270.385569
## iter 10 value 216.166446
## iter 20 value 185.522611
## iter 30 value 173.564105
## iter 40 value 155.099991
## iter 50 value 139.144087
## iter 60 value 129.026928
## iter 70 value 119.723323
## iter 80 value 107.451068
## iter 90 value 91.210481
## iter 100 value 78.142450
## final value 78.142450
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 205.902789
## iter 20 value 154.466750
## iter 30 value 139.653249
## iter 40 value 119.238914
## iter 50 value 105.406380
## iter 60 value 88.117632
## iter 70 value 81.629122
## iter 80 value 67.715729
## iter 90 value 50.488608
## iter 100 value 41.579569
## final value 41.579569
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 205.902819
## iter 20 value 154.467845
## iter 30 value 139.654432
## iter 40 value 119.255897
## iter 50 value 105.473859
## iter 60 value 88.376053
## iter 70 value 81.017059
## iter 80 value 67.879342
## iter 90 value 53.401341
## iter 100 value 44.545720
## final value 44.545720
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 205.902789
## iter 20 value 154.466751
## iter 30 value 139.653250
## iter 40 value 119.238931
## iter 50 value 105.406448
## iter 60 value 88.117893
## iter 70 value 81.629666
## iter 80 value 67.717719
## iter 90 value 50.492750
## iter 100 value 41.591961
## final value 41.591961

```

```

## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 207.828724
## iter 20 value 176.130978
## iter 30 value 150.697453
## iter 40 value 133.751248
## iter 50 value 116.398121
## iter 60 value 103.381280
## iter 70 value 93.375389
## iter 80 value 83.585598
## iter 90 value 68.466644
## iter 100 value 57.109247
## final value 57.109247
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 207.828758
## iter 20 value 176.131477
## iter 30 value 150.702178
## iter 40 value 133.771356
## iter 50 value 116.447071
## iter 60 value 103.577894
## iter 70 value 94.171941
## iter 80 value 84.466346
## iter 90 value 69.755066
## iter 100 value 59.272431
## final value 59.272431
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 207.828724
## iter 20 value 176.130978
## iter 30 value 150.697458
## iter 40 value 133.751268
## iter 50 value 116.398170
## iter 60 value 103.381476
## iter 70 value 93.376113
## iter 80 value 83.586754
## iter 90 value 68.469243
## iter 100 value 57.171045
## final value 57.171045
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 197.459887
## iter 20 value 171.807460
## iter 30 value 162.216908
## iter 40 value 145.682362
## iter 50 value 131.191764
## iter 60 value 116.528722
## iter 70 value 99.801864
## iter 80 value 87.048354
## iter 90 value 71.433299

```



```

## iter 100 value 62.605663
## final value 62.605663
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 197.459939
## iter 20 value 171.808111
## iter 30 value 162.218561
## iter 40 value 145.695424
## iter 50 value 131.200207
## iter 60 value 117.099548
## iter 70 value 100.169194
## iter 80 value 87.031378
## iter 90 value 72.315875
## iter 100 value 65.278252
## final value 65.278252
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 197.459888
## iter 20 value 171.807461
## iter 30 value 162.216910
## iter 40 value 145.682375
## iter 50 value 131.191771
## iter 60 value 116.529329
## iter 70 value 99.791085
## iter 80 value 87.059637
## iter 90 value 71.418633
## iter 100 value 61.334656
## final value 61.334656
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 202.702743
## iter 20 value 159.830659
## iter 30 value 139.978162
## iter 40 value 118.274196
## iter 50 value 100.017759
## iter 60 value 88.746993
## iter 70 value 83.800188
## iter 80 value 71.801326
## iter 90 value 59.118625
## iter 100 value 44.540729
## final value 44.540729
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 202.702828
## iter 20 value 159.831649
## iter 30 value 139.981279
## iter 40 value 118.291279
## iter 50 value 100.096491
## iter 60 value 88.961493
## iter 70 value 84.107769

```

```

## iter 80 value 73.071035
## iter 90 value 62.452530
## iter 100 value 50.377832
## final value 50.377832
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 202.702743
## iter 20 value 159.830660
## iter 30 value 139.978165
## iter 40 value 118.274213
## iter 50 value 100.017838
## iter 60 value 88.747209
## iter 70 value 83.800488
## iter 80 value 71.802620
## iter 90 value 59.120887
## iter 100 value 44.563505
## final value 44.563505
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 202.431157
## iter 20 value 175.359213
## iter 30 value 160.382800
## iter 40 value 140.003843
## iter 50 value 124.504878
## iter 60 value 111.647835
## iter 70 value 99.352902
## iter 80 value 87.131942
## iter 90 value 70.093546
## iter 100 value 56.313193
## final value 56.313193
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 202.431192
## iter 20 value 175.359877
## iter 30 value 160.385404
## iter 40 value 140.021855
## iter 50 value 124.572136
## iter 60 value 111.845314
## iter 70 value 99.830156
## iter 80 value 86.696370
## iter 90 value 71.285503
## iter 100 value 59.653013
## final value 59.653013
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 202.431157
## iter 20 value 175.359214
## iter 30 value 160.382803
## iter 40 value 140.003861
## iter 50 value 124.504945

```

```

## iter 60 value 111.648030
## iter 70 value 99.353384
## iter 80 value 87.135619
## iter 90 value 70.097516
## iter 100 value 56.327647
## final value 56.327647
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 201.371805
## iter 20 value 166.031971
## iter 30 value 150.075932
## iter 40 value 135.132305
## iter 50 value 120.592996
## iter 60 value 104.861049
## iter 70 value 94.760756
## iter 80 value 86.209074
## iter 90 value 70.479689
## iter 100 value 59.293608
## final value 59.293608
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 201.371834
## iter 20 value 166.032587
## iter 30 value 150.078439
## iter 40 value 135.145979
## iter 50 value 120.647244
## iter 60 value 105.100395
## iter 70 value 95.630193
## iter 80 value 86.629358
## iter 90 value 73.769241
## iter 100 value 62.418039
## final value 62.418039
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 201.371805
## iter 20 value 166.031971
## iter 30 value 150.075934
## iter 40 value 135.132319
## iter 50 value 120.593051
## iter 60 value 104.861290
## iter 70 value 94.761635
## iter 80 value 86.204322
## iter 90 value 70.477907
## iter 100 value 59.380210
## final value 59.380210
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 193.758552
## iter 20 value 161.466052
## iter 30 value 143.149426

```

```

## iter 40 value 119.906737
## iter 50 value 105.972213
## iter 60 value 94.038716
## iter 70 value 86.611103
## iter 80 value 76.999325
## iter 90 value 64.304603
## iter 100 value 49.108593
## final value 49.108593
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 193.758575
## iter 20 value 161.466759
## iter 30 value 143.152818
## iter 40 value 119.925067
## iter 50 value 106.063601
## iter 60 value 94.476557
## iter 70 value 88.125313
## iter 80 value 79.353741
## iter 90 value 68.383915
## iter 100 value 55.778418
## final value 55.778418
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 193.758552
## iter 20 value 161.466053
## iter 30 value 143.149429
## iter 40 value 119.906755
## iter 50 value 105.972298
## iter 60 value 94.039119
## iter 70 value 86.612063
## iter 80 value 77.024633
## iter 90 value 64.294495
## iter 100 value 48.946686
## final value 48.946686
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 205.700241
## iter 20 value 172.245031
## iter 30 value 161.350362
## iter 40 value 145.949562
## iter 50 value 134.646862
## iter 60 value 123.431117
## iter 70 value 115.427646
## iter 80 value 107.575646
## iter 90 value 93.359323
## iter 100 value 78.496595
## final value 78.496595
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 205.700275

```

```

## iter 20 value 172.245662
## iter 30 value 161.352088
## iter 40 value 145.960485
## iter 50 value 134.688955
## iter 60 value 123.574379
## iter 70 value 115.743901
## iter 80 value 108.252481
## iter 90 value 95.287892
## iter 100 value 84.379872
## final value 84.379872
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 205.700241
## iter 20 value 172.245032
## iter 30 value 161.350363
## iter 40 value 145.949573
## iter 50 value 134.646905
## iter 60 value 123.431261
## iter 70 value 115.427971
## iter 80 value 107.576296
## iter 90 value 93.361320
## iter 100 value 78.502007
## final value 78.502007
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 219.760490
## iter 20 value 187.124426
## iter 30 value 169.999087
## iter 40 value 150.988583
## iter 50 value 138.413835
## iter 60 value 123.009829
## iter 70 value 109.478001
## iter 80 value 93.806014
## iter 90 value 75.758745
## iter 100 value 58.395941
## final value 58.395941
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 219.760529
## iter 20 value 187.125144
## iter 30 value 170.001985
## iter 40 value 151.002233
## iter 50 value 138.477224
## iter 60 value 123.272620
## iter 70 value 111.151892
## iter 80 value 92.955032
## iter 90 value 77.276762
## iter 100 value 64.227998
## final value 64.227998
## stopped after 100 iterations
## # weights: 285 (224 variable)

```

```

## initial value 270.385569
## iter 10 value 219.760490
## iter 20 value 187.124427
## iter 30 value 169.999090
## iter 40 value 150.988596
## iter 50 value 138.413899
## iter 60 value 123.010094
## iter 70 value 109.478646
## iter 80 value 93.807518
## iter 90 value 75.762081
## iter 100 value 58.410349
## final value 58.410349
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 202.083849
## iter 20 value 171.218444
## iter 30 value 148.002738
## iter 40 value 131.483341
## iter 50 value 119.899625
## iter 60 value 101.885103
## iter 70 value 88.167837
## iter 80 value 77.579263
## iter 90 value 67.403824
## iter 100 value 54.845989
## final value 54.845989
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 202.083917
## iter 20 value 171.219158
## iter 30 value 148.006594
## iter 40 value 131.494731
## iter 50 value 119.947547
## iter 60 value 102.164080
## iter 70 value 88.909884
## iter 80 value 80.246572
## iter 90 value 70.884610
## iter 100 value 62.437067
## final value 62.437067
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 202.083849
## iter 20 value 171.218445
## iter 30 value 148.002742
## iter 40 value 131.483352
## iter 50 value 119.899672
## iter 60 value 101.885379
## iter 70 value 88.168693
## iter 80 value 77.580995
## iter 90 value 67.407601
## iter 100 value 54.895975
## final value 54.895975

```

```

## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter   10 value 210.893574
## iter   20 value 176.968481
## iter   30 value 157.621487
## iter   40 value 142.056823
## iter   50 value 121.819529
## iter   60 value 111.531475
## iter   70 value 100.966866
## iter   80 value  88.183437
## iter   90 value  73.319758
## iter  100 value  58.907600
## final   value  58.907600
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter   10 value 210.893616
## iter   20 value 176.969104
## iter   30 value 157.624217
## iter   40 value 142.070829
## iter   50 value 121.894347
## iter   60 value 111.706223
## iter   70 value 101.934582
## iter   80 value  88.907037
## iter   90 value  74.853558
## iter  100 value  62.676642
## final   value  62.676642
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter   10 value 210.893574
## iter   20 value 176.968482
## iter   30 value 157.621490
## iter   40 value 142.056837
## iter   50 value 121.819604
## iter   60 value 111.531651
## iter   70 value 100.967339
## iter   80 value  88.182492
## iter   90 value  73.321790
## iter  100 value  58.923957
## final   value  58.923957
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter   10 value 215.234341
## iter   20 value 187.495648
## iter   30 value 156.884090
## iter   40 value 126.627495
## iter   50 value 108.092316
## iter   60 value  93.655427
## iter   70 value  82.956918
## iter   80 value  70.727102
## iter   90 value  54.497158

```

```

## iter 100 value 38.707672
## final value 38.707672
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 215.234405
## iter 20 value 187.496165
## iter 30 value 156.890467
## iter 40 value 126.670853
## iter 50 value 108.136155
## iter 60 value 93.903580
## iter 70 value 82.571252
## iter 80 value 71.747868
## iter 90 value 60.863266
## iter 100 value 46.666754
## final value 46.666754
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 215.234341
## iter 20 value 187.495649
## iter 30 value 156.884096
## iter 40 value 126.627538
## iter 50 value 108.092360
## iter 60 value 93.655673
## iter 70 value 82.958070
## iter 80 value 70.733976
## iter 90 value 54.499717
## iter 100 value 38.184015
## final value 38.184015
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 206.360036
## iter 20 value 172.487861
## iter 30 value 155.021658
## iter 40 value 136.335948
## iter 50 value 119.177256
## iter 60 value 110.476952
## iter 70 value 103.090165
## iter 80 value 95.068108
## iter 90 value 79.645467
## iter 100 value 66.656328
## final value 66.656328
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 206.360068
## iter 20 value 172.488629
## iter 30 value 155.023441
## iter 40 value 136.350067
## iter 50 value 119.240533
## iter 60 value 110.611543
## iter 70 value 103.346467

```



```

## iter 80 value 95.453720
## iter 90 value 81.089286
## iter 100 value 71.797990
## final value 71.797990
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 206.360036
## iter 20 value 172.487862
## iter 30 value 155.021660
## iter 40 value 136.335963
## iter 50 value 119.177320
## iter 60 value 110.477087
## iter 70 value 103.090422
## iter 80 value 95.067651
## iter 90 value 79.646829
## iter 100 value 66.657618
## final value 66.657618
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 220.183939
## iter 20 value 185.813346
## iter 30 value 166.002072
## iter 40 value 145.689793
## iter 50 value 129.203808
## iter 60 value 112.779371
## iter 70 value 101.156987
## iter 80 value 89.181144
## iter 90 value 71.341174
## iter 100 value 60.270997
## final value 60.270997
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 220.183976
## iter 20 value 185.813893
## iter 30 value 166.005036
## iter 40 value 145.706838
## iter 50 value 129.260881
## iter 60 value 113.034823
## iter 70 value 101.614811
## iter 80 value 90.462165
## iter 90 value 74.152421
## iter 100 value 64.434659
## final value 64.434659
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 220.183939
## iter 20 value 185.813346
## iter 30 value 166.002075
## iter 40 value 145.689810
## iter 50 value 129.203865

```

```
## iter 60 value 112.779628
## iter 70 value 101.157452
## iter 80 value 89.182589
## iter 90 value 71.344062
## iter 100 value 60.255059
## final value 60.255059
## stopped after 100 iterations
## # weights: 285 (224 variable)
## initial value 270.385569
## iter 10 value 213.786480
## iter 20 value 183.820116
## iter 30 value 175.610350
## iter 40 value 166.416809
## iter 50 value 156.687239
## iter 60 value 150.770066
## iter 70 value 141.604223
## iter 80 value 132.373761
## iter 90 value 117.371155
## iter 100 value 107.188847
## final value 107.188847
## stopped after 100 iterations
```

```
summary(log.tune)
```

```
## Call:
## nnet::multinom(formula = .outcome ~ ., data = dat, decay = param$decay,
## direction = "forward", k = ..2)
##
## Coefficients:
## (Intercept)      age      viewcat      setting      viewenc      prebody
## 2 -0.10430024 0.2129859 -0.34982789 0.04138759 -0.09653652 -0.3325268
## 3 0.06293638 0.5017950 -0.86594527 0.17242433 0.34030478 -1.5850149
## 4 -0.11348294 0.1084882 -0.09580164 -0.05144057 -0.28198446 -0.2926012
## 5 -0.04232544 0.1268965 -0.00157604 -0.04201233 -0.16079544 0.2624026
##      prelet      preform      prenumb      prerelat      preclasf 'age:viewcat'
## 2 1.07548218 -1.19154198 -0.1355582 0.01218500 -0.8281805 0.01101840
## 3 -0.09390197 -0.31628259 0.9186298 0.31343738 -0.5633445 -0.02328669
## 4 -0.30935254 0.01800312 -0.2184987 -0.26198492 0.9754936 0.08728599
## 5 0.02435272 0.22175132 -0.6841873 0.08376244 0.2796019 0.01044005
## 'age:setting' 'age:viewenc' 'age:prebody' 'age:prelet' 'age:preform'
## 2 -0.1950952 0.15083709 -0.015740783 0.003796499 0.03454772
## 3 -0.2402140 0.05406668 0.006006701 0.002664593 0.04427898
## 4 0.0370847 0.03283911 -0.013181506 0.011476032 0.01067502
## 5 -0.1834337 0.06846291 -0.011654324 0.010500400 0.00502330
## 'age:prenumb' 'age:prerelat' 'age:preclasf' 'viewcat:setting'
## 2 0.002748548 0.017421552 -0.03449594 -0.7157048
## 3 -0.021179041 0.006862094 -0.02046393 1.9296577
## 4 0.005381244 0.002834208 -0.03462003 -1.3304799
## 5 0.004025159 0.021167701 -0.01529625 -0.6193439
## 'viewcat:viewenc' 'viewcat:prebody' 'viewcat:prelet' 'viewcat:preform'
## 2 -0.6267928 0.14158801 -0.18278208 -0.33356833
## 3 -0.6451057 0.15032040 -0.19113344 -0.19616855
## 4 -0.7600318 0.02100516 -0.06168443 -0.09204344
## 5 -0.7298604 -0.01705615 -0.08760215 -0.00448432
```

```

## 'viewcat:prenumb' 'viewcat:prerelat' 'viewcat:preclasf' 'setting:viewenc'
## 2 0.036088877 0.14586964 0.23145567 0.07630395
## 3 0.021174552 0.31836093 -0.07370337 0.86832568
## 4 0.003855981 0.07887976 -0.10100460 -0.63499806
## 5 0.087689197 0.24193609 -0.07174626 0.37153187
## 'setting:prebody' 'setting:prelet' 'setting:preform' 'setting:prenumb'
## 2 0.469099240 -0.4122878 -0.13051934 0.05284150
## 3 0.135607010 -0.1971371 0.01884259 0.05295298
## 4 -0.006161835 -0.2960129 -0.10589631 0.26263061
## 5 0.421579113 -0.2513299 0.15172993 0.04497442
## 'setting:prerelat' 'setting:preclasf' 'viewenc:prebody' 'viewenc:prelet'
## 2 0.9158782 -0.20327566 -0.1732281313 -0.15753436
## 3 -0.6874554 0.77873656 -0.1003194119 0.22031347
## 4 0.2901539 -0.09297208 0.0008472876 0.10954771
## 5 0.1997050 -0.02496983 -0.4660637395 0.08286859
## 'viewenc:preform' 'viewenc:prenumb' 'viewenc:prerelat' 'viewenc:preclasf'
## 2 -0.56985394 0.03247080 -1.3095372 1.4372910
## 3 -0.78356022 0.06169254 -0.8061659 0.8328746
## 4 -0.01496193 -0.09635533 -0.2666694 0.3122020
## 5 -0.17168716 0.20628325 -0.4768000 0.5294078
## 'prebody:prelet' 'prebody:preform' 'prebody:prenumb' 'prebody:prerelat'
## 2 -0.026728574 0.02310599 -0.004690487 0.06567582
## 3 0.008919958 -0.07757222 0.004168160 0.08221147
## 4 0.002594866 -0.06621870 -0.005876983 0.08031019
## 5 -0.018649963 -0.05937411 -0.031267621 0.12347594
## 'prebody:preclasf' 'prelet:preform' 'prelet:prenumb' 'prelet:prerelat'
## 2 0.01621294 0.053767578 -0.024222497 0.010990911
## 3 0.04294225 0.037711752 -0.028066953 0.054014235
## 4 0.07572520 0.003193047 -0.011355172 -0.003240511
## 5 0.04845629 0.023606964 0.002600023 -0.039934799
## 'prelet:preclasf' 'preform:prenumb' 'preform:prerelat' 'preform:preclasf'
## 2 0.040880193 -0.06146975 0.0001794692 0.072813218
## 3 -0.006339726 0.02589022 -0.0338563963 -0.005385365
## 4 0.029870629 0.01796772 0.0091114657 0.046634386
## 5 0.028841972 0.03215846 -0.0931836919 0.036711837
## 'prenumb:prerelat' 'prenumb:preclasf' 'prerelat:preclasf'
## 2 0.0347272941 0.034704048 -0.2159680
## 3 -0.0023133262 0.006896269 -0.1384417
## 4 -0.0002585178 -0.019663447 -0.1506193
## 5 0.0255919065 -0.001445600 -0.2108050
##
## Std. Errors:
## (Intercept) age viewcat setting viewenc prebody prelet
## 2 0.01769491 0.2281321 0.04463295 0.017978327 0.02743229 0.2795415 0.2451148
## 3 0.02272730 0.1783334 0.06424362 0.029933348 0.03521339 0.2644438 0.4033421
## 4 0.02199825 0.1887900 0.06092779 0.024090342 0.03376304 0.3572217 0.4088392
## 5 0.01018045 0.2010834 0.03038656 0.009964914 0.01420608 0.1814051 0.2567150
## preform prenumb prerelat preclasf 'age:viewcat' 'age:setting'
## 2 0.11178013 0.3364181 0.09881117 0.13663278 0.05635717 0.09530280
## 3 0.13921911 0.3950844 0.09883397 0.14861756 0.05043176 0.09138367
## 4 0.13220239 0.3274458 0.11556801 0.14712060 0.04885287 0.08840051
## 5 0.09857628 0.1968719 0.07393768 0.09452461 0.06297351 0.11500433
## 'age:viewenc' 'age:prebody' 'age:prelet' 'age:preform' 'age:prenumb'
## 2 0.11304152 0.01433104 0.01356935 0.02786496 0.01497572

```

## 3	0.09915620	0.01349342	0.01315086	0.02339626	0.01438330
## 4	0.09331245	0.01429374	0.01221907	0.02392057	0.01476449
## 5	0.13061367	0.01728838	0.01324726	0.02935786	0.01581603
##	'age:prerelat' 'age:preclasf' 'viewcat:setting' 'viewcat:viewenc'				
## 2	0.02584354	0.02741556	0.2490771	0.3332642	
## 3	0.02480077	0.02372367	0.2847775	0.2908848	
## 4	0.02586148	0.02541847	0.3029041	0.4216808	
## 5	0.02754718	0.02878072	0.1607059	0.1494858	
##	'viewcat:prebody' 'viewcat:prelet' 'viewcat:preform' 'viewcat:prenumb'				
## 2	0.1108576	0.11755013	0.2010431	0.09697412	
## 3	0.1219053	0.11054466	0.1733514	0.09255908	
## 4	0.1107068	0.09731512	0.1698175	0.08853306	
## 5	0.1380371	0.12158710	0.2223183	0.11578095	
##	'viewcat:prerelat' 'viewcat:preclasf' 'setting:viewenc' 'setting:prebody'				
## 2	0.2751360	0.1629227	0.08493856	0.2330639	
## 3	0.2601977	0.1541121	0.07712831	0.2437100	
## 4	0.2304470	0.1370595	0.13297704	0.2560144	
## 5	0.2974449	0.2124611	0.06689098	0.3039591	
##	'setting:prelet' 'setting:preform' 'setting:prenumb' 'setting:prerelat'				
## 2	0.2149093	0.3573498	0.2038061	0.3706735	
## 3	0.2632529	0.3247801	0.2250007	0.3562895	
## 4	0.2180408	0.3425432	0.2068494	0.3963333	
## 5	0.2570749	0.3889281	0.2517049	0.2415947	
##	'setting:preclasf' 'viewenc:prebody' 'viewenc:prelet' 'viewenc:preform'				
## 2	0.3342676	0.2457295	0.1839863	0.3807850	
## 3	0.3125047	0.2560498	0.1967347	0.3360116	
## 4	0.3433573	0.2228863	0.1694260	0.3106407	
## 5	0.3701965	0.3151663	0.2098595	0.4171969	
##	'viewenc:prenumb' 'viewenc:prerelat' 'viewenc:preclasf' 'prebody:prelet'				
## 2	0.1837119	0.4035093	0.3503429	0.02038602	
## 3	0.2189024	0.3392222	0.3549874	0.02341377	
## 4	0.1679850	0.4089991	0.3004770	0.01934844	
## 5	0.2426103	0.3200753	0.3722014	0.02845440	
##	'prebody:preform' 'prebody:prenumb' 'prebody:prerelat' 'prebody:preclasf'				
## 2	0.05337533	0.02326317	0.06436399	0.05175746	
## 3	0.05680423	0.02309044	0.06277950	0.04906464	
## 4	0.05059798	0.02129793	0.06010049	0.04585516	
## 5	0.06802325	0.02925413	0.07559607	0.05945588	
##	'prelet:preform' 'prelet:prenumb' 'prelet:prerelat' 'prelet:preclasf'				
## 2	0.04333995	0.01443954	0.04451862	0.03412912	
## 3	0.04561577	0.01585042	0.04955802	0.03562827	
## 4	0.04016587	0.01239328	0.04068016	0.02845603	
## 5	0.04602906	0.01454398	0.05396960	0.03432086	
##	'preform:prenumb' 'preform:prerelat' 'preform:preclasf' 'prenumb:prerelat'				
## 2	0.04078930	0.10960277	0.07055708	0.04544892	
## 3	0.04170952	0.10147248	0.06993119	0.04374414	
## 4	0.03378745	0.08337231	0.05951540	0.04023916	
## 5	0.04886538	0.11943772	0.08905010	0.05140125	
##	'prenumb:preclasf' 'prerelat:preclasf'				
## 2	0.03153069	0.10242770			
## 3	0.02978874	0.09538739			
## 4	0.03086232	0.09338443			
## 5	0.04096462	0.11263783			
##					

```
## Residual Deviance: 214.3777
## AIC: 662.3777
```

```
tabs2 <- table(true=test.log[, "site"],
               pred=predict(log.tune,newdata=test.log))
confusionMatrix(tabs2)
```

```
## Confusion Matrix and Statistics
```

```
##
```

```
##      pred
```

```
## true  1  2  3  4  5
```

```
##      1  4  1  5  4  4
```

```
##      2  3  6  7  1  1
```

```
##      3  3  0 16  0  0
```

```
##      4  0  1  6  4  1
```

```
##      5  1  1  3  0  0
```

```
##
```

```
## Overall Statistics
```

```
##
```

```
##              Accuracy : 0.4167
```

```
##              95% CI : (0.3015, 0.5389)
```

```
##      No Information Rate : 0.5139
```

```
##      P-Value [Acc > NIR] : 0.961676
```

```
##
```

```
##              Kappa : 0.2408
```

```
##
```

```
## McNemar's Test P-Value : 0.006843
```

```
##
```

```
## Statistics by Class:
```

```
##
```

```
##              Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
```

```
## Sensitivity      0.36364  0.66667  0.4324  0.44444  0.00000
```

```
## Specificity      0.77049  0.80952  0.9143  0.87302  0.92424
```

```
## Pos Pred Value   0.22222  0.33333  0.8421  0.33333  0.00000
```

```
## Neg Pred Value   0.87037  0.94444  0.6038  0.91667  0.91045
```

```
## Prevalence       0.15278  0.12500  0.5139  0.12500  0.08333
```

```
## Detection Rate   0.05556  0.08333  0.2222  0.05556  0.00000
```

```
## Detection Prevalence 0.25000  0.25000  0.2639  0.16667  0.06944
```

```
## Balanced Accuracy 0.56706  0.73810  0.6734  0.65873  0.46212
```

Questions for OH:

should we transform regular?

Both linear and radial kernels never output predictions for 4 & 5?

polynomial kernel? Which variables to give polynomial terms

use PCA to perform feature selection?

feature selections for SVM in general?

how to interpret the confusion matrix tables for SVM & Trees

How to interpret the importance variance for multiclass classification

interpretations about the dataset, using the bad performance of the classifiers