Read Data

11/23/2021

```
library(foreign)
library(tidyverse)
```

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

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.1     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(e1071)
library(tree)
```

```
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

```
library(nnet)
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
library(randomForest)
```

```
## 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
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

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

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

```
## Loading required package: MBA

## Loading required package: gstat

## Loading required package: automap

## Loading required package: sp
```

```r
library(scales)
```

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

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

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

```r
library(kableExtra)
```

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

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

```r
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

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

## Loaded glmnet 4.1-2
```

```r
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

```r
head(sesame)
```

```
##   rownames id site sex age viewcat setting viewenc prebody prelet preform
## 1        1  1    1   1  66       1       2       1      16     23      12
## 2        2  2    1   2  67       3       2       1      30     26       9
## 3        3  3    1   1  56       3       2       2      22     14       9
## 4        4  4    1   1  49       1       2       2      23     11      10
## 5        5  5    1   1  69       4       2       2      32     47      15
## 6        6  6    1   2  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 decribe 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") +
```

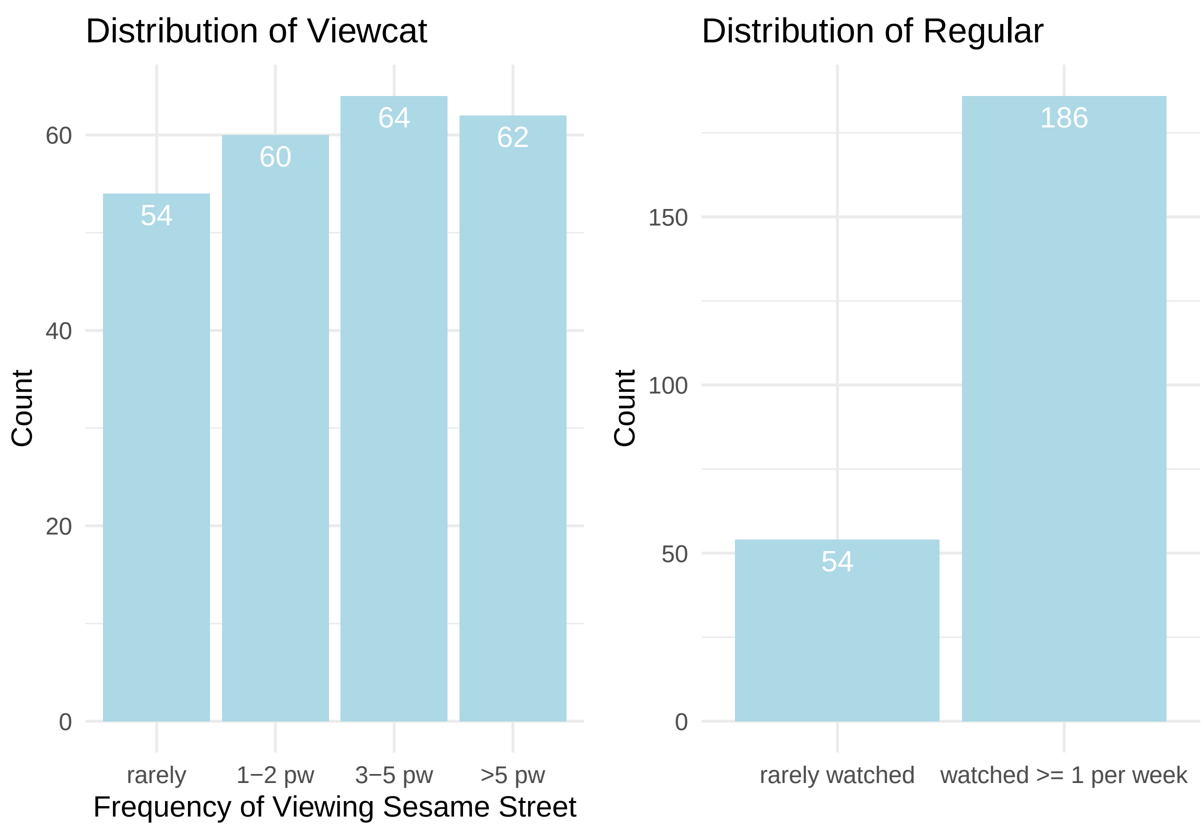
```r
  theme_minimal()

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 distrubtions 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
```

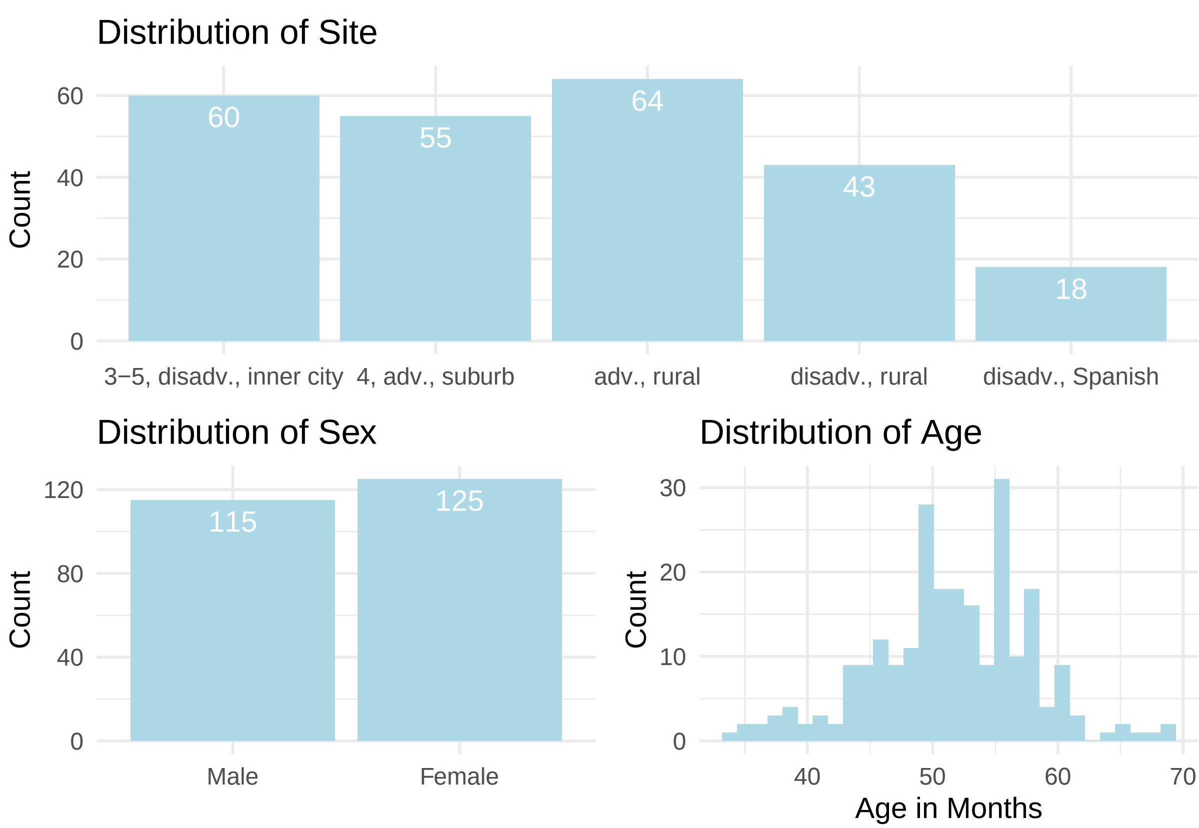


Both of these variables are categorical. On the left, viewcat appears to have a roughly uniform distribution, with "rarely" having the least amount of children and 3-5 times per week having the most (the range is only 10 children, so all of the bars are relatively close in height). For the variable regular, 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., rura
  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
```

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

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

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

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

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

```r
# 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
```

```r
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
```

```r
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
```

```r
AIC(lin.mod2.full)
```

```
## [1] 1004.281
```

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

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

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

```r
# 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
```

```r
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(numbDiff ~ (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
```

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

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

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

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

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

```r
# 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)
```

```r
# 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
```

```r
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
```

```r
AIC(lin.mod5.full)
```

```
## [1] 988.3598
```

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

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

```
## [1] 18.94131
```

```r
# 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
```

```r
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
```

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

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

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

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

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

```r
# 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

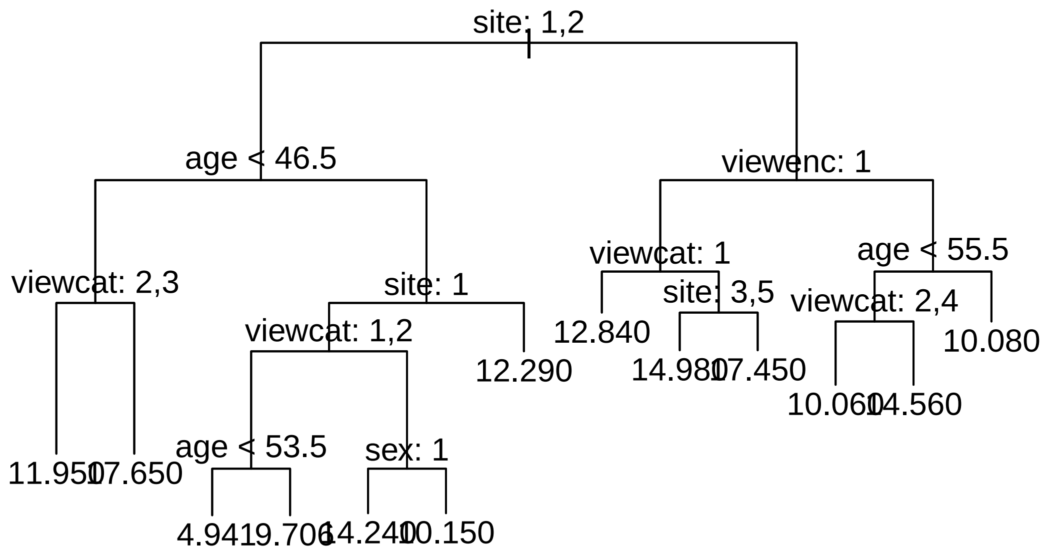
-
-

Model 1

```
set.seed(1)

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

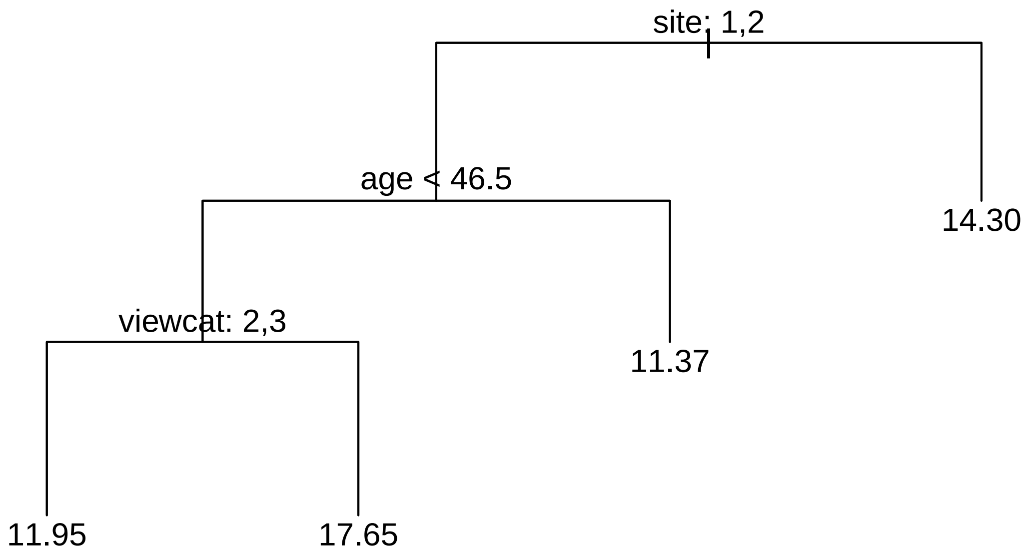
```
##
## Regression tree:
## tree(formula = bodyDiff ~ site + sex + age + viewcat + setting +
##     viewenc, data = sesame.q1, subset = train)
## Variables actually used in tree construction:
## [1] "site"    "age"     "viewcat" "sex"     "viewenc"
## Number of terminal nodes:  13
## Residual mean 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")
```


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




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

```
##
## Regression tree:
## snip.tree(tree = reg.tree.1, nodes = c(5L, 3L))
## Variables actually used in tree construction:
## [1] "site"    "age"     "viewcat"
## Number of terminal nodes:  4
## Residual mean deviance:  17.39 = 2852 / 164
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -12.5400  -2.5470   0.2536   0.0000   2.4650  15.7000
```

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

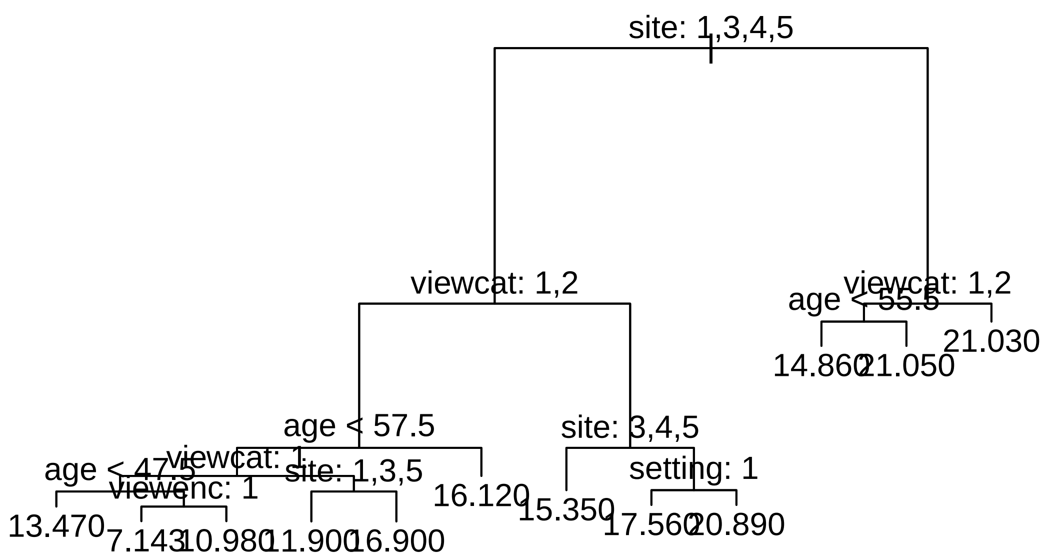
```
## [1] 20.60159
```

Model 2

```r
set.seed(1)

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

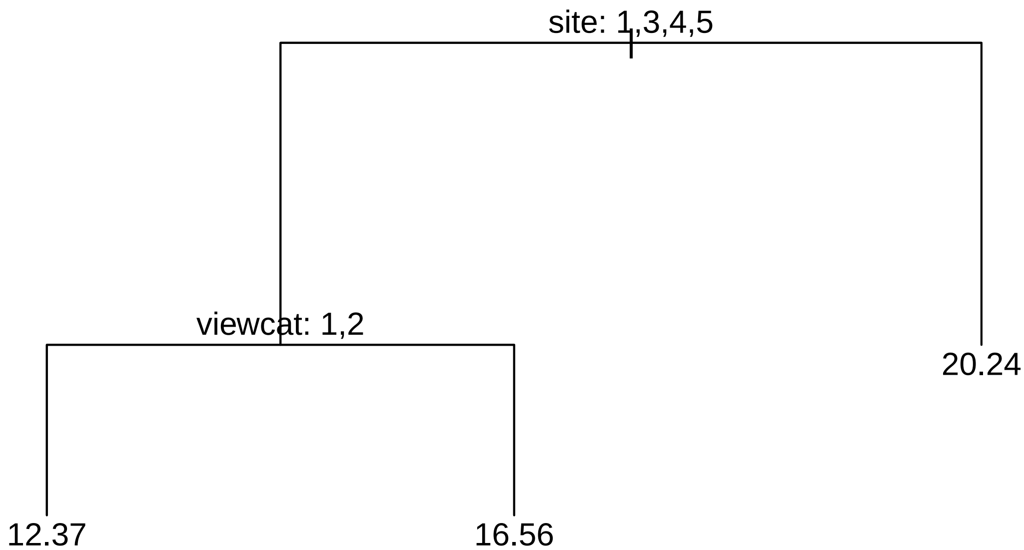
```
##
## Regression tree:
## tree(formula = letDiff ~ site + sex + age + viewcat + setting +
##     viewenc, data = sesame.q1, subset = train)
## Variables actually used in tree construction:
## [1] "site"    "viewcat" "age"     "viewenc" "setting"
## Number of terminal nodes:  12
## Residual mean deviance:  18.63 = 2906 / 156
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -17.560  -2.430   0.000   0.000   2.765   9.587
```

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




```r
cv.reg.tree.2  <- cv.tree(reg.tree.2)
plot(cv.reg.tree.2$size, cv.reg.tree.2$dev, type = "b")
```


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




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

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

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

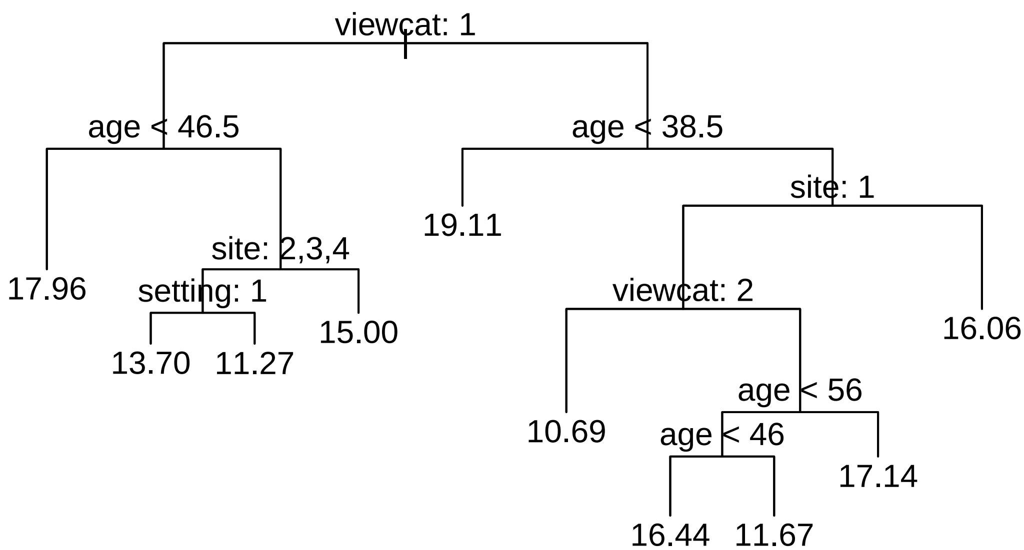
Model 3

```r
set.seed(1)

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

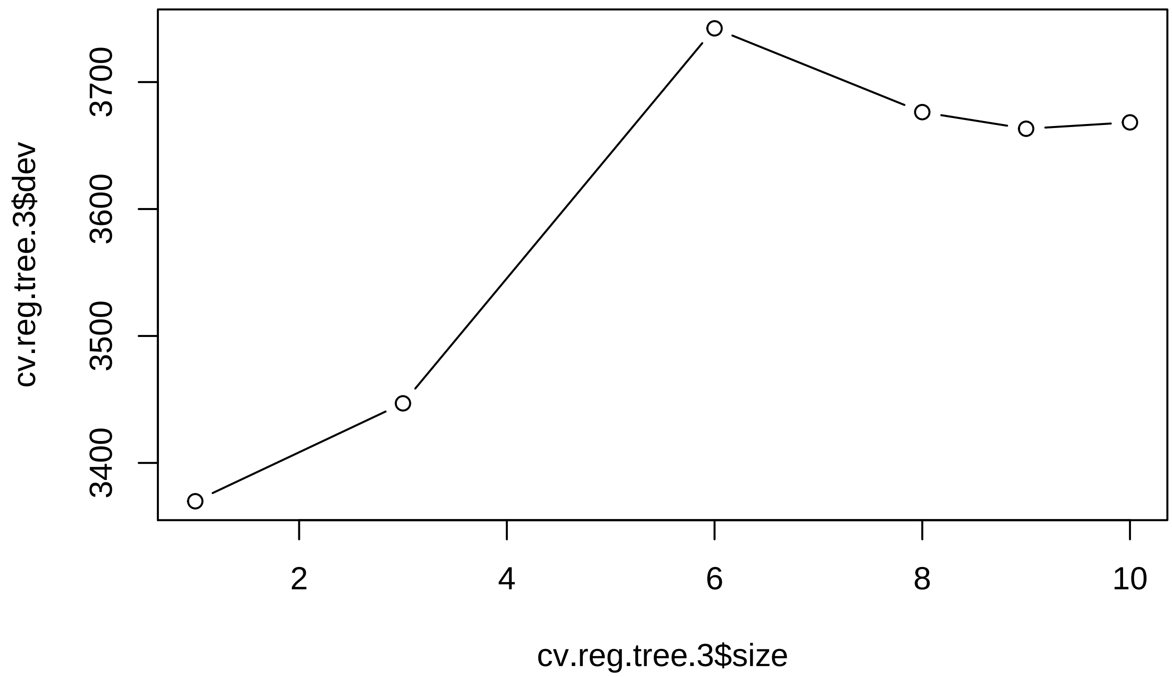
```
##
## Regression tree:
## tree(formula = formDiff ~ site + sex + age + viewcat + setting +
##     viewenc, data = sesame.q1, subset = train)
## Variables actually used in tree construction:
## [1] "viewcat" "age"     "site"    "setting"
## Number of terminal nodes:  10
## Residual mean deviance: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

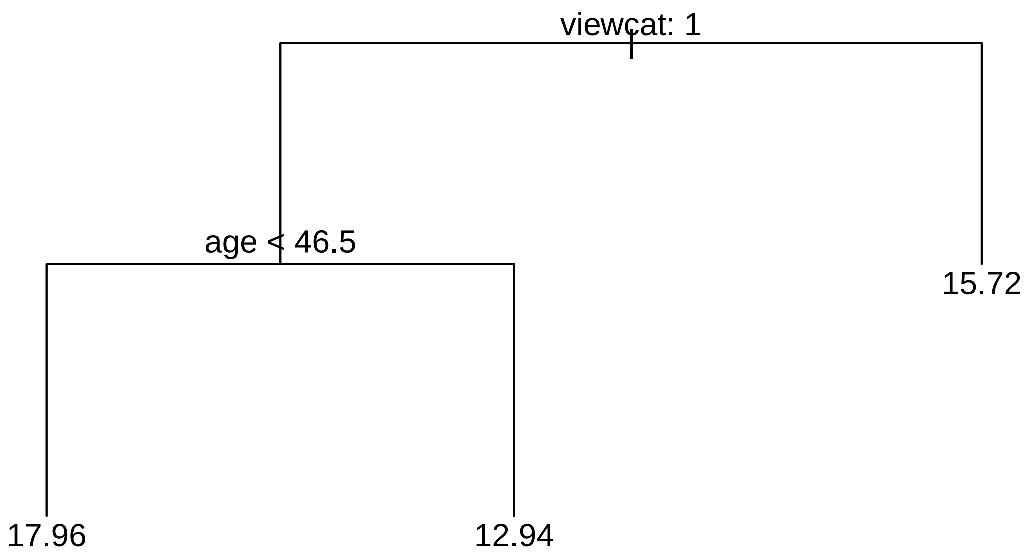
```
plot(reg.tree.3)
text(reg.tree.3, pretty = 0)
```




```
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)
```

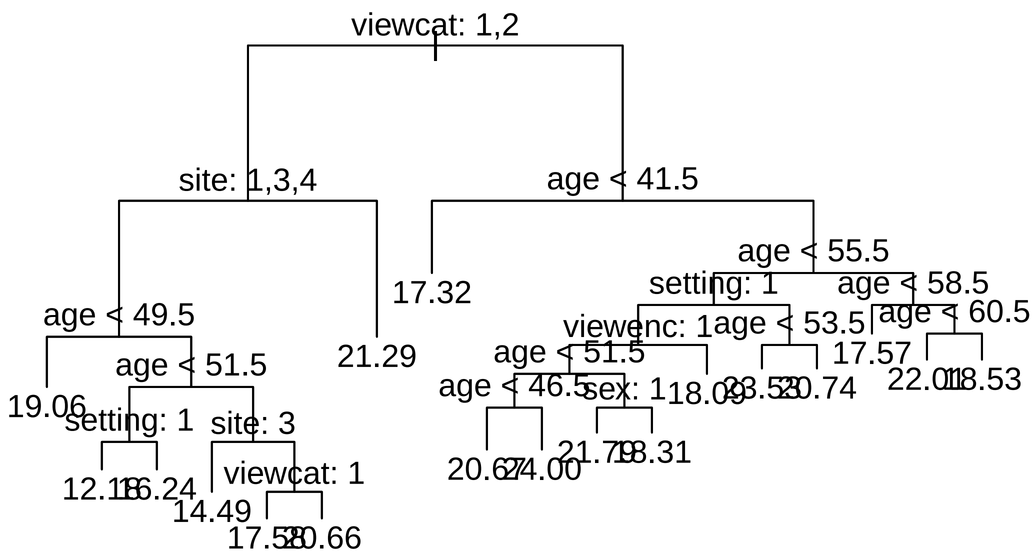
```
## [1] 14.92273
```

Model 4

```
set.seed(1)

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

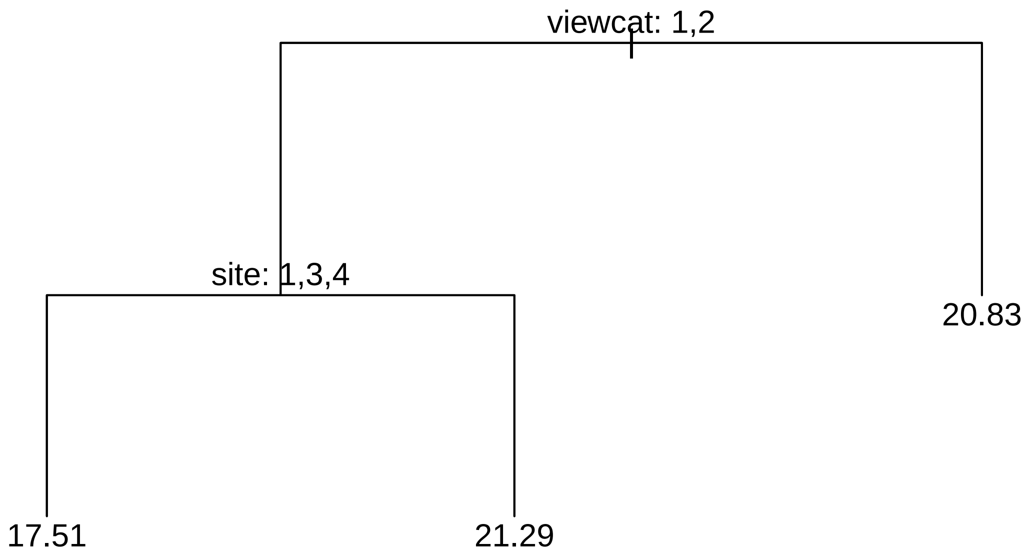
```
##
## Regression tree:
## tree(formula = numbDiff ~ site + sex + age + viewcat + setting +
##     viewenc, data = sesame.q1, subset = train)
## Number of terminal nodes:  18
## Residual mean deviance:  13.64 = 2046 / 150
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.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

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




```r
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
```

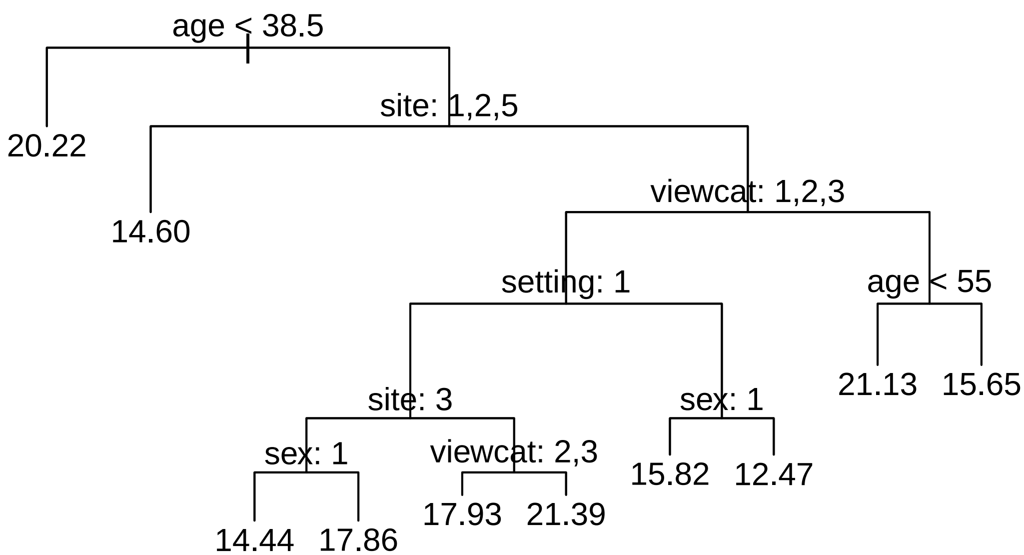
Model 5

```
set.seed(1)

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

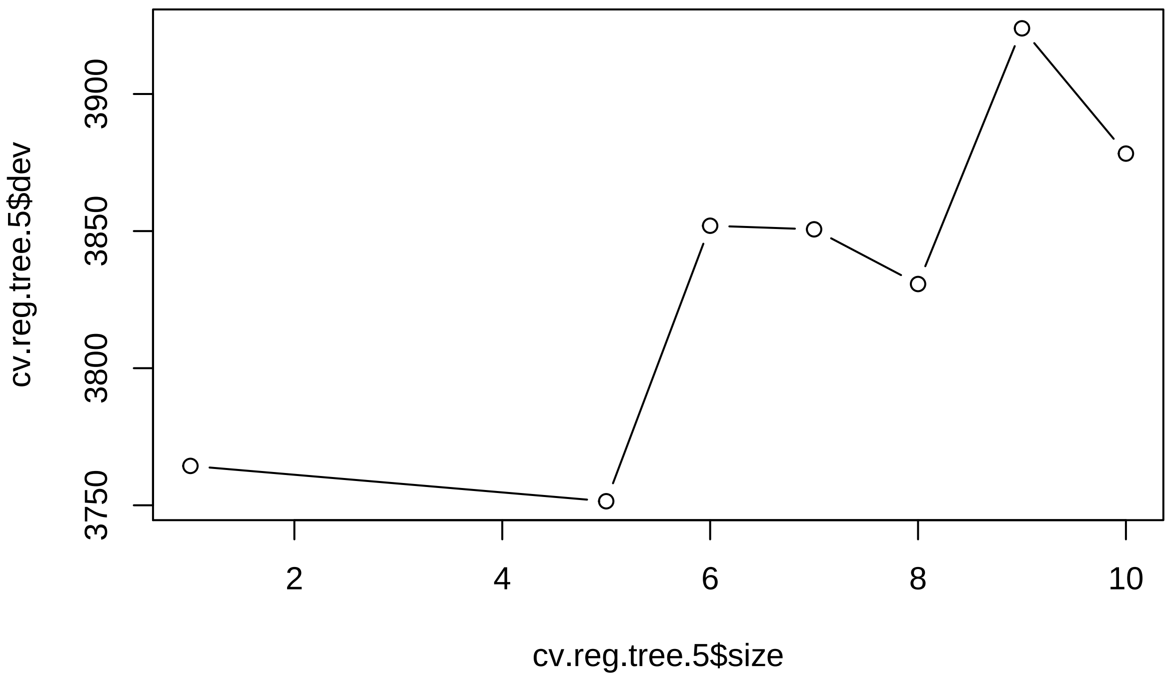
```
##
## Regression tree:
## tree(formula = relatDiff ~ site + sex + age + viewcat + setting +
##     viewenc, data = sesame.q1, subset = train)
## Variables actually used in tree construction:
## [1] "age"     "site"    "viewcat" "setting" "sex"
## Number of terminal nodes:  10
## Residual mean deviance: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

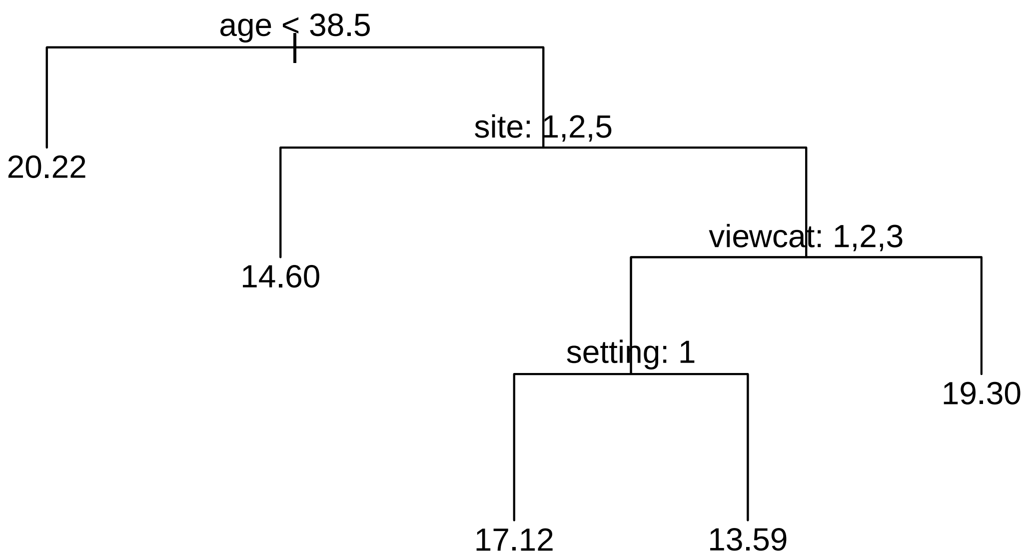
```
plot(reg.tree.5)
text(reg.tree.5, pretty = 0)
```




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




```r
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
```

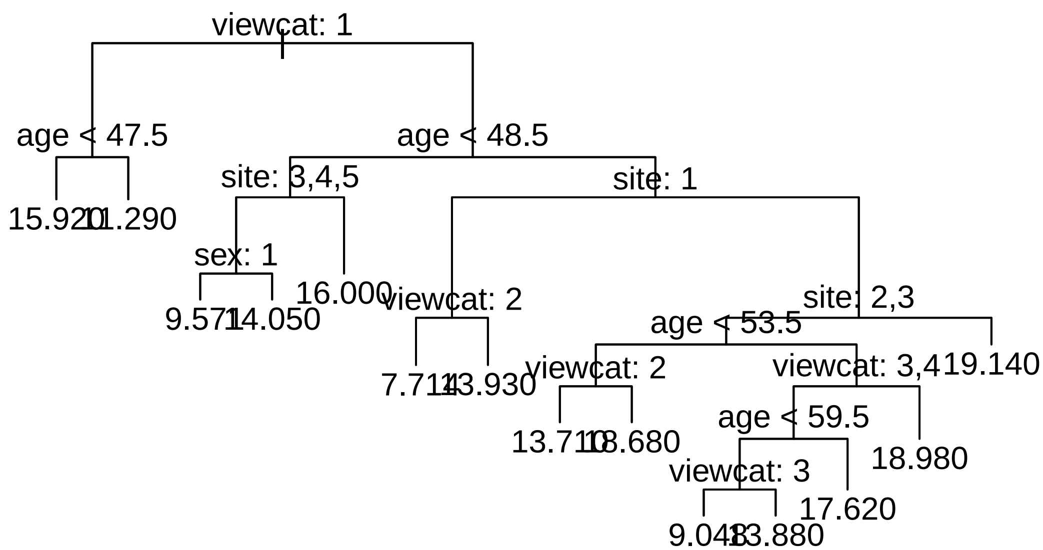
Model 6

```
set.seed(1)

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

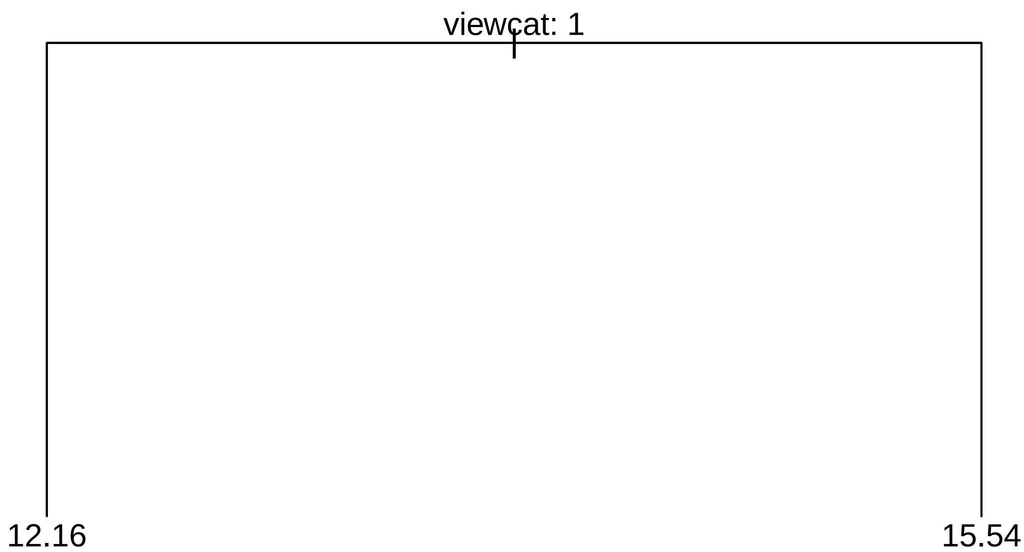
```
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -16.00000  -3.17300   0.01852   0.00000   3.71400  13.10000
```

```
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 Sc
#
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.Regression.Test.MSE	Ridge.Regression.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_pcl
                    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
```

```r
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
```

```r
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
```

```r
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 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 3 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 4 4 5 5 5 5 5
## Levels: 1 2 3 4 5
```

Radial kernel improves prediction on class 1.

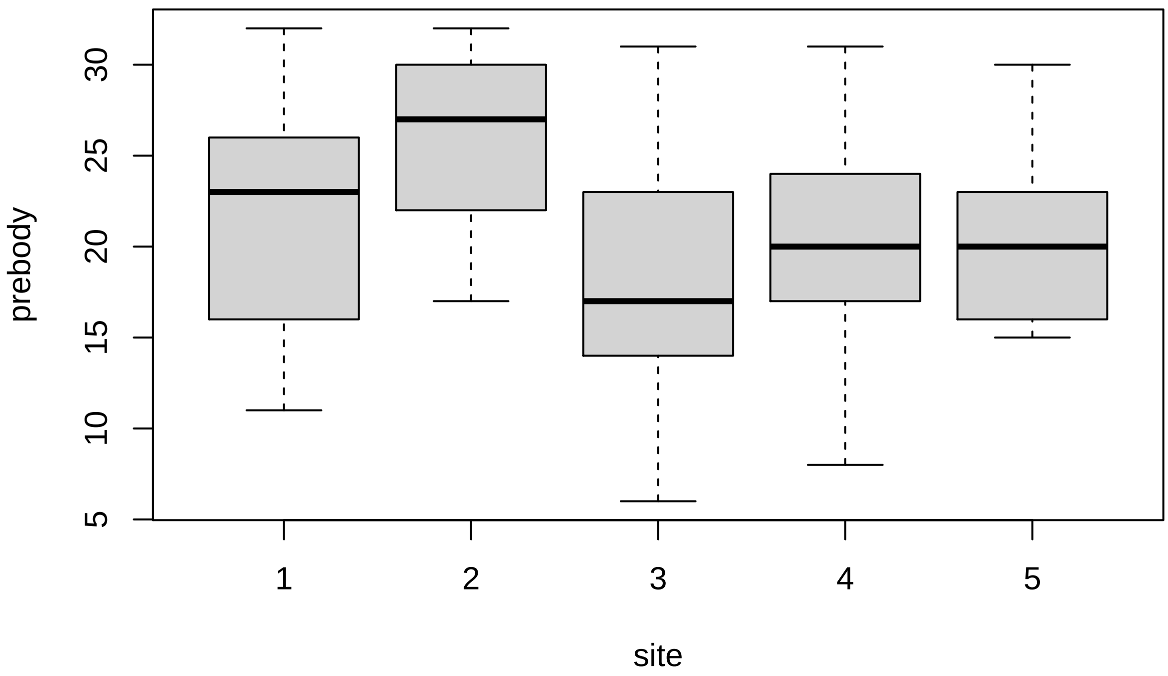
RBF slightly improved after standardizing? (it seems slightly more likely to predict on class 1.) thought, simpler models still retain the same performance (arguably better) sd_age+sd_pBod+sd_plet. But we are still not getting any prediction on class 4 & 5.

After assign class weights using this formula:

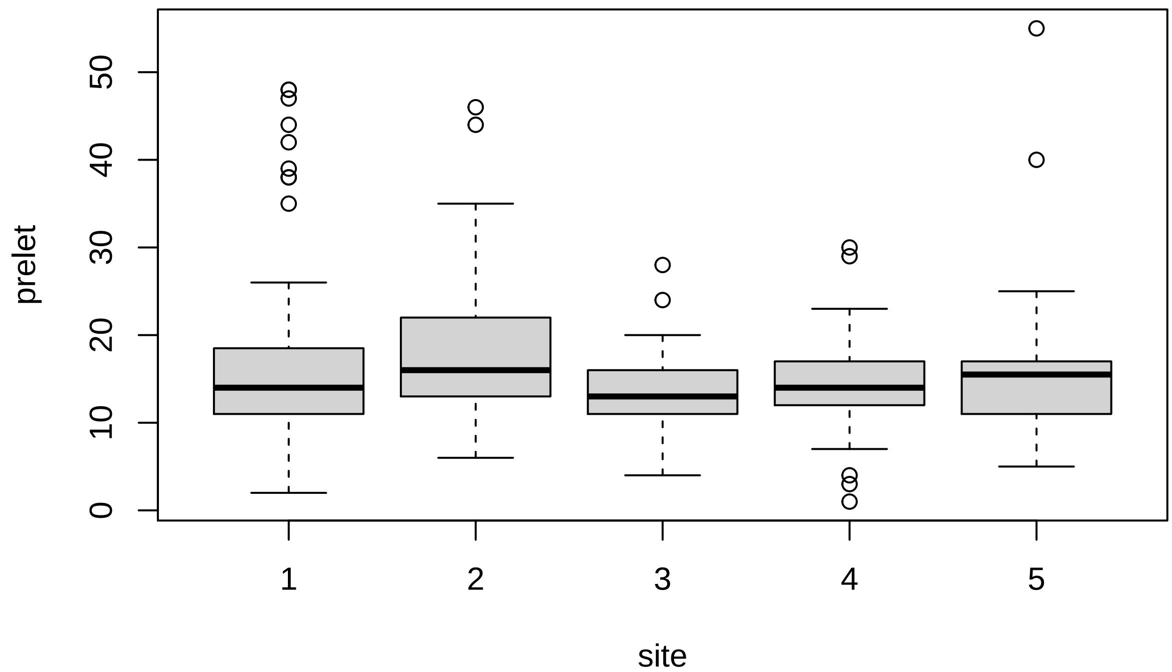
$$w_j = \frac{n}{kn_j},\ \text{n is total number of data points, k is number of classes}$$

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).

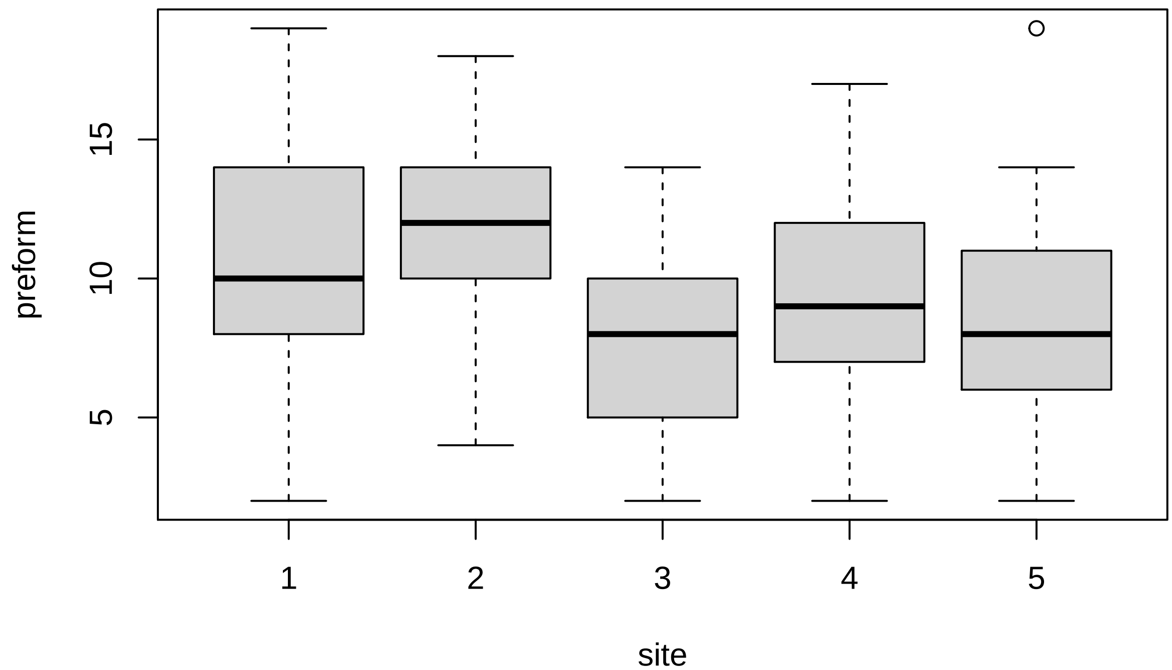
```
# trying to do more EDA to see if anything explains why the data is not linearly separable
boxplot(prebody~site, data=sesame)
```



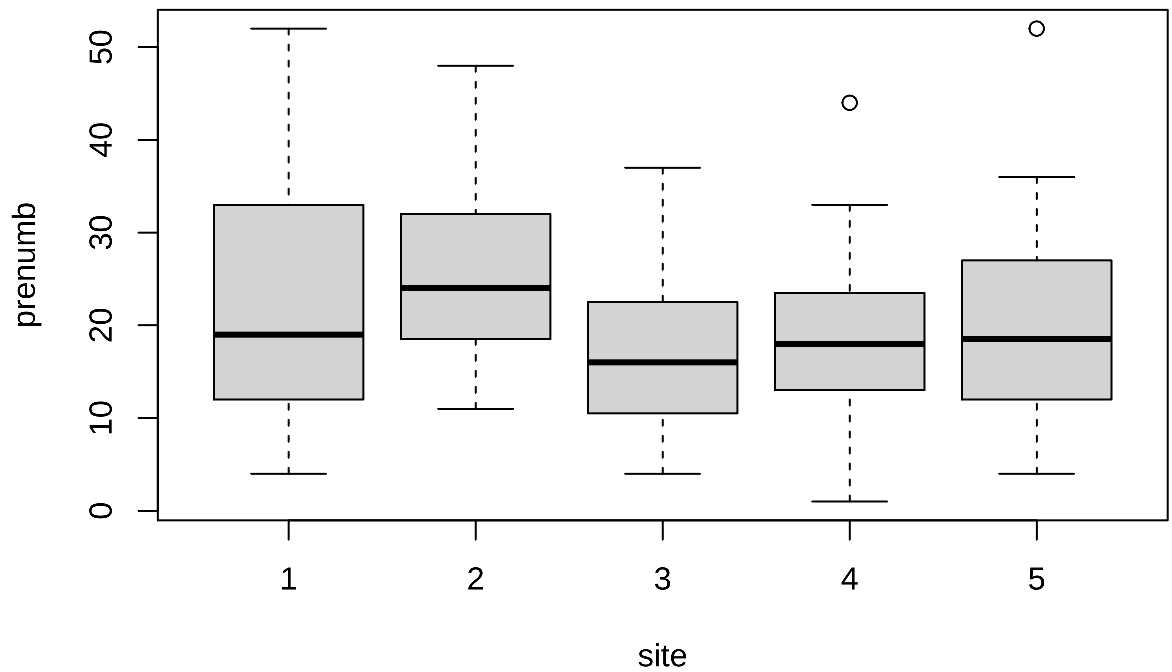

```
boxplot(prelet~site, data=sesame)
```



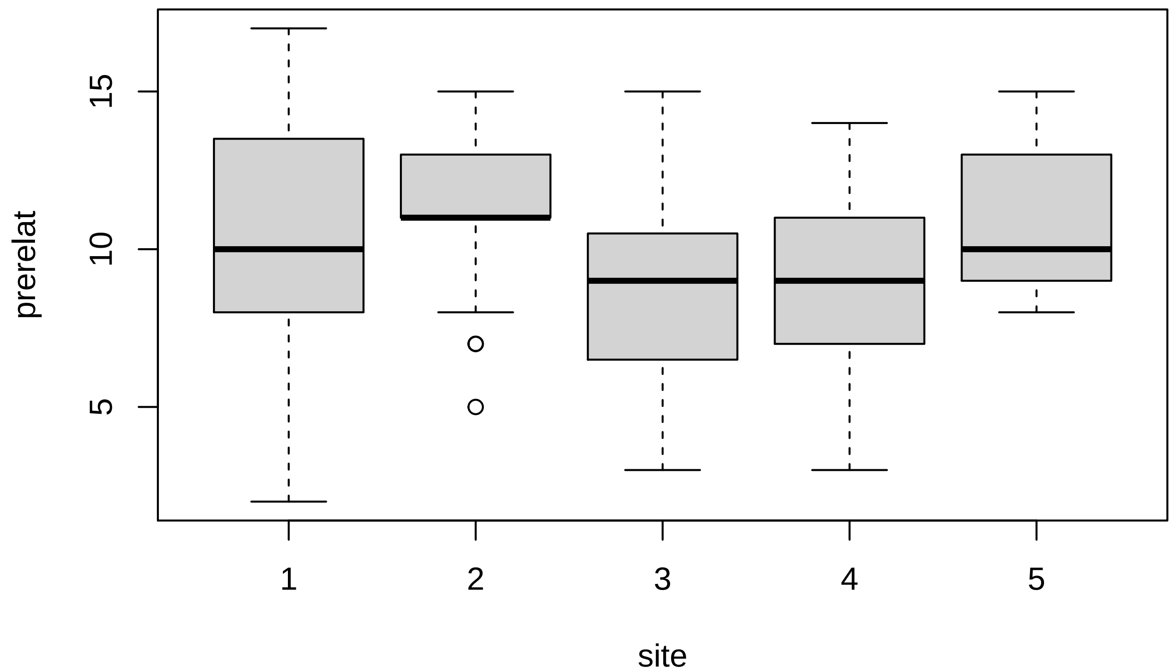

```
boxplot(preform~site, data=sesame)
```



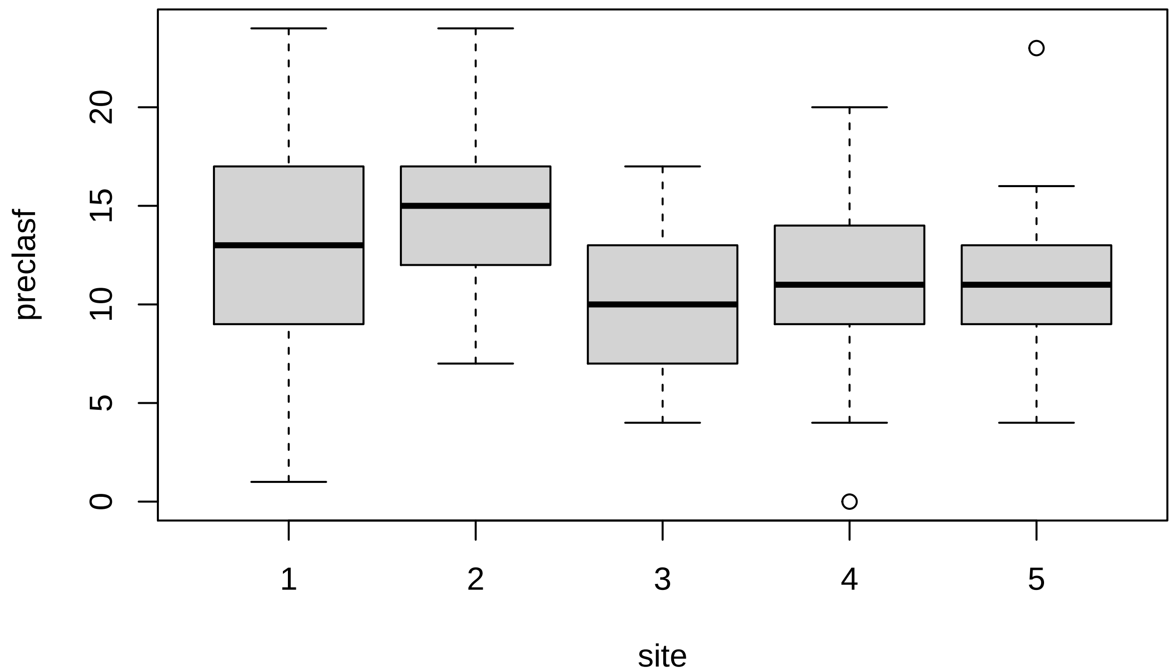

```r
boxplot(prenumb~site, data=sesame)
```



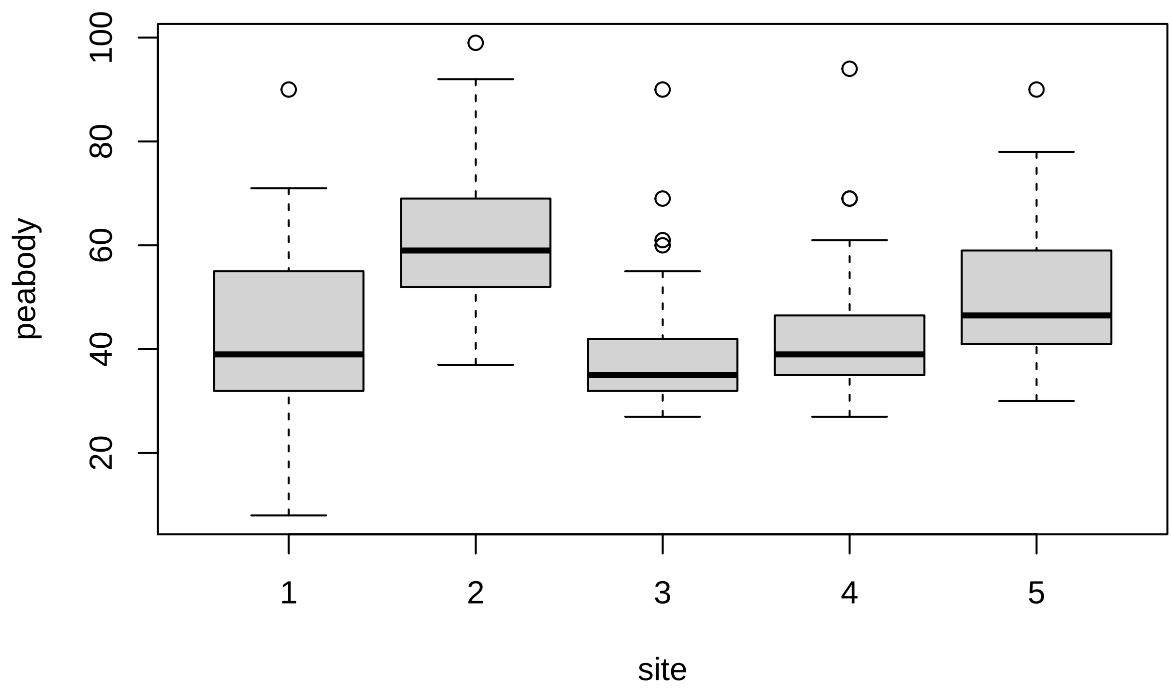

```
boxplot(prerelat~site, data=sesame)
```




```
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", "prebody", "prelet","preform", "pren

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)
```

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

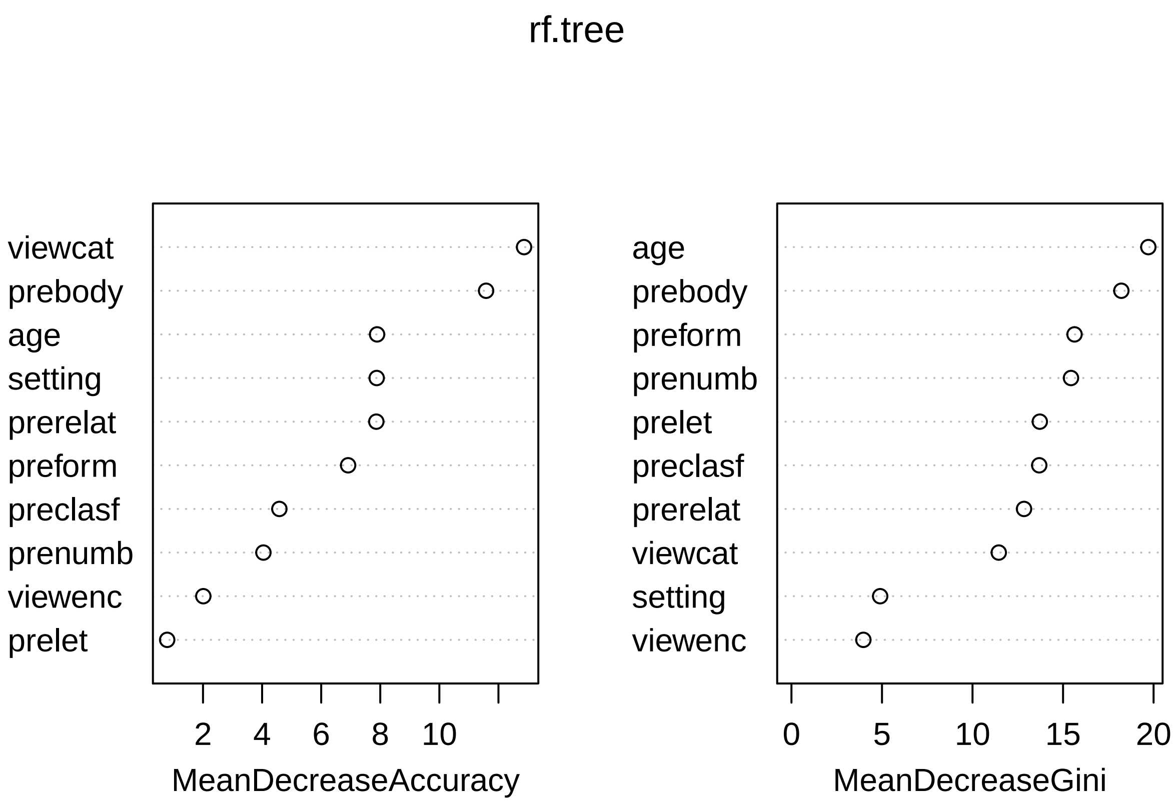
```
##                  1          2          3          4          5
## age      11.9546291  5.9587749 -3.5567843  1.2194447 -2.9305154
## viewcat   3.4297393  7.3567844  2.3303342 13.4298675  2.5255798
## setting   5.3521907 -2.2017505  4.8249895  4.4857572  6.7085517
## viewenc   2.2903099  1.5495633  2.6331821 -1.0388159 -2.2631215
## prebody  -3.7294779 11.8553418 10.2295785  4.0752640 -1.7308649
## prelet    4.0595901 -0.7462124 -0.9117786 -0.5283485 -2.8915518
## preform   0.5943075  3.2989200 10.5406588 -2.8661046  2.0996114
## prenumb   5.0870239  0.8052775  1.2400743  2.4001741 -1.0130831
## prerelat  7.5022090  6.4354777  2.5537622 -0.1100140 -1.3464452
## preclasf  5.2204065  2.6343969  3.3735235 -3.8870094  0.2778988
##          MeanDecreaseAccuracy MeanDecreaseGini
## age                 7.8926243        19.709380
## viewcat            12.8667032        11.449491
## setting             7.8790749         4.898302
## viewenc             2.0101660         3.971704
## prebody            11.5822476        18.219834
## prelet              0.7877462        13.714974
## preform             6.9114180        15.633358
## prenumb             4.0446841        15.437768
## prerelat            7.8663167        12.844868
## preclasf            4.5850016        13.684573
```

```r
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  6  3  7  2  0
##    2  4  9  3  2  0
##    3  5  0 12  2  0
##    4  1  1  5  5  0
##    5  1  0  2  2  0
##
## Overall Statistics
##
##                Accuracy : 0.4444
##                  95% CI : (0.3272, 0.5664)
##     No Information Rate : 0.4028
##     P-Value [Acc > NIR] : 0.2725
##
##                   Kappa : 0.2685
##
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.35294   0.6923   0.4138  0.38462       NA
## Specificity           0.78182   0.8475   0.8372  0.88136  0.93056
## Pos Pred Value        0.33333   0.5000   0.6316  0.41667       NA
## Neg Pred Value        0.79630   0.9259   0.6792  0.86667       NA
## Prevalence            0.23611   0.1806   0.4028  0.18056  0.00000
## Detection Rate        0.08333   0.1250   0.1667  0.06944  0.00000
## Detection Prevalence  0.25000   0.2500   0.2639  0.16667  0.06944
## Balanced Accuracy     0.56738   0.7699   0.6255  0.63299       NA
```

```
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 $k = 5$.

```r
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
```

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

importance(rf.tree)
```

```
##                  1          2         3          4         5
## age       7.054870  0.2028937 -3.361101  2.8057841  9.315800
## viewcat   4.469857  6.1615700  2.727869 16.8763289 19.746867
## setting   7.871285 -0.7871503  2.930196 13.4637866 29.376697
## viewenc   1.915990  0.7650218  2.402900  2.7222992  3.810101
## prebody  -1.690110 11.1161914  2.832516  3.1220234 12.965107
## prelet    5.924971 -1.6168977  0.331860  3.4483562  7.196834
## preform   3.381609 -0.9026224  9.200003 -0.6041287 21.170468
## prenumb   6.748919 -2.6427548  2.090979  1.1802431  7.266489
## prerelat  8.340101  0.2831294  6.635648  3.6858341 15.883281
## preclasf  5.641388  4.2338261  2.950919 -4.4058643 17.349840
##          MeanDecreaseAccuracy MeanDecreaseGini
## age                  7.490781        16.660187
## viewcat             23.782113        13.165949
## setting             24.921609         8.343210
## viewenc              5.055138         2.426533
## prebody             12.784029        16.915283
## prelet               7.630572        14.100819
## preform             17.948977        15.433474
## prenumb              6.770577        14.879388
## prerelat            17.327568        16.552560
## preclasf            14.507151        15.842830
```

```r
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  3  4  7  3  1
##    2  2 10  2  2  2
##    3  5  1  9  3  1
##    4  1  1  0  6  4
##    5  1  0  2  1  1
##
## Overall Statistics
##
##                Accuracy : 0.4028
##                  95% CI : (0.2888, 0.525)
##     No Information Rate : 0.2778
##     P-Value [Acc > NIR] : 0.01481
##
##                   Kappa : 0.2402
##
##  Mcnemar's Test P-Value : 0.45821
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.25000   0.6250   0.4500  0.40000  0.11111
## Specificity           0.75000   0.8571   0.8077  0.89474  0.93651
## Pos Pred Value        0.16667   0.5556   0.4737  0.50000  0.20000
## Neg Pred Value        0.83333   0.8889   0.7925  0.85000  0.88060
## Prevalence            0.16667   0.2222   0.2778  0.20833  0.12500
## Detection Rate        0.04167   0.1389   0.1250  0.08333  0.01389
## Detection Prevalence  0.25000   0.2500   0.2639  0.16667  0.06944
## Balanced Accuracy     0.50000   0.7411   0.6288  0.64737  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.

```r
# 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,]
```

```r
# 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

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

tree.features <- c("site", "age", "viewcat", "setting", "viewenc","prebody", "prelet","preform", "prenu

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
```

```r
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="backward",
                  k=log(3562))
```

```
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 195.918067
## iter  20 value 157.877929
## iter  30 value 148.130728
## iter  40 value 133.755458
## iter  50 value 116.729947
## iter  60 value 106.240160
## iter  70 value 95.437633
## iter  80 value 85.441772
## iter  90 value 70.333381
## iter 100 value 49.959588
## final  value 49.959588
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 195.918103
## iter  20 value 157.878542
## iter  30 value 148.132409
## iter  40 value 133.770830
## iter  50 value 116.776390
## iter  60 value 106.556531
## iter  70 value 95.864798
## iter  80 value 86.199932
## iter  90 value 70.083263
## iter 100 value 53.064084
## final  value 53.064084
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 195.918067
## iter  20 value 157.877930
## iter  30 value 148.130730
## iter  40 value 133.755473
## iter  50 value 116.729994
## iter  60 value 106.240399
## iter  70 value 95.437887
## iter  80 value 85.442513
## iter  90 value 70.339030
## iter 100 value 49.965276
## final  value 49.965276
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 208.790330
## iter  20 value 191.082246
```

```
## iter  30 value 173.845872
## iter  40 value 154.801242
## iter  50 value 138.864452
## iter  60 value 128.792324
## iter  70 value 119.300330
## iter  80 value 108.974328
## iter  90 value 86.190873
## iter 100 value 69.137407
## final  value 69.137407
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 208.790349
## iter  20 value 191.082538
## iter  30 value 173.849084
## iter  40 value 154.814442
## iter  50 value 138.943921
## iter  60 value 128.938845
## iter  70 value 119.547476
## iter  80 value 109.586335
## iter  90 value 89.792465
## iter 100 value 72.504725
## final  value 72.504725
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 208.790330
## iter  20 value 191.082246
## iter  30 value 173.845875
## iter  40 value 154.801255
## iter  50 value 138.864531
## iter  60 value 128.792471
## iter  70 value 119.300589
## iter  80 value 108.974873
## iter  90 value 86.184167
## iter 100 value 69.205620
## final  value 69.205620
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 213.526273
## iter  20 value 176.169119
## iter  30 value 147.481119
## iter  40 value 131.765483
## iter  50 value 116.584517
## iter  60 value 102.275443
## iter  70 value 92.524977
## iter  80 value 82.091999
## iter  90 value 64.511986
## iter 100 value 51.728257
## final  value 51.728257
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
```

```
## iter  10 value 213.526297
## iter  20 value 176.169853
## iter  30 value 147.485811
## iter  40 value 131.779656
## iter  50 value 116.644675
## iter  60 value 102.478808
## iter  70 value 93.017120
## iter  80 value 82.058425
## iter  90 value 65.976429
## iter 100 value 55.744081
## final  value 55.744081
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 213.526273
## iter  20 value 176.169120
## iter  30 value 147.481124
## iter  40 value 131.765497
## iter  50 value 116.584577
## iter  60 value 102.275647
## iter  70 value 92.525455
## iter  80 value 82.093815
## iter  90 value 64.511363
## iter 100 value 51.757440
## final  value 51.757440
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 199.968628
## iter  20 value 164.778357
## iter  30 value 147.486730
## iter  40 value 132.775185
## iter  50 value 118.693053
## iter  60 value 107.335318
## iter  70 value 99.197283
## iter  80 value 88.669865
## iter  90 value 72.619265
## iter 100 value 58.207510
## final  value 58.207510
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 199.968668
## iter  20 value 164.778884
## iter  30 value 147.489259
## iter  40 value 132.786193
## iter  50 value 118.744533
## iter  60 value 107.489377
## iter  70 value 99.478313
## iter  80 value 89.426808
## iter  90 value 74.487706
## iter 100 value 62.306036
## final  value 62.306036
## stopped after 100 iterations
```

```
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 199.968628
## iter  20 value 164.778358
## iter  30 value 147.486732
## iter  40 value 132.775196
## iter  50 value 118.693105
## iter  60 value 107.335473
## iter  70 value 99.197568
## iter  80 value 88.670642
## iter  90 value 72.621275
## iter 100 value 58.211546
## final  value 58.211546
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 198.382413
## iter  20 value 154.611690
## iter  30 value 145.295659
## iter  40 value 132.885148
## iter  50 value 118.698321
## iter  60 value 107.763939
## iter  70 value 96.783555
## iter  80 value 84.830939
## iter  90 value 67.183087
## iter 100 value 50.962528
## final  value 50.962528
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 198.382473
## iter  20 value 154.612554
## iter  30 value 145.297521
## iter  40 value 132.891154
## iter  50 value 118.744215
## iter  60 value 107.891514
## iter  70 value 97.165527
## iter  80 value 87.187237
## iter  90 value 72.477637
## iter 100 value 59.208817
## final  value 59.208817
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 198.382413
## iter  20 value 154.611691
## iter  30 value 145.295661
## iter  40 value 132.885154
## iter  50 value 118.698367
## iter  60 value 107.764057
## iter  70 value 96.785255
## iter  80 value 84.834411
## iter  90 value 67.187461
## iter 100 value 50.975741
```

```
## final  value 50.975741
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 200.286263
## iter  20 value 152.079929
## iter  30 value 131.259946
## iter  40 value 111.422752
## iter  50 value 97.639535
## iter  60 value 83.919965
## iter  70 value 74.255686
## iter  80 value 64.544934
## iter  90 value 48.855356
## iter 100 value 34.983099
## final  value 34.983099
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 200.286288
## iter  20 value 152.080921
## iter  30 value 131.262668
## iter  40 value 111.438640
## iter  50 value 97.685483
## iter  60 value 84.091434
## iter  70 value 74.583903
## iter  80 value 65.178513
## iter  90 value 50.749124
## iter 100 value 37.616964
## final  value 37.616964
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 200.286263
## iter  20 value 152.079930
## iter  30 value 131.259948
## iter  40 value 111.422768
## iter  50 value 97.639581
## iter  60 value 83.920137
## iter  70 value 74.256004
## iter  80 value 64.545459
## iter  90 value 48.857386
## iter 100 value 34.975010
## final  value 34.975010
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 203.668217
## iter  20 value 166.098903
## iter  30 value 146.400034
## iter  40 value 131.113118
## iter  50 value 116.240727
## iter  60 value 106.960662
## iter  70 value 97.176330
## iter  80 value 87.190796
```

```
## iter  90 value 68.802616
## iter 100 value 56.862340
## final  value 56.862340
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 203.668245
## iter  20 value 166.099644
## iter  30 value 146.403239
## iter  40 value 131.124803
## iter  50 value 116.311056
## iter  60 value 107.183047
## iter  70 value 97.615611
## iter  80 value 88.125302
## iter  90 value 71.362618
## iter 100 value 61.009529
## final  value 61.009529
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 203.668217
## iter  20 value 166.098904
## iter  30 value 146.400037
## iter  40 value 131.113130
## iter  50 value 116.240798
## iter  60 value 106.960880
## iter  70 value 97.176773
## iter  80 value 87.191671
## iter  90 value 68.805357
## iter 100 value 56.867432
## final  value 56.867432
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 193.255856
## iter  20 value 169.566226
## iter  30 value 161.547616
## iter  40 value 150.922408
## iter  50 value 135.852364
## iter  60 value 128.169972
## iter  70 value 118.142808
## iter  80 value 106.923262
## iter  90 value 83.813527
## iter 100 value 67.152695
## final  value 67.152695
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 193.255891
## iter  20 value 169.566602
## iter  30 value 161.548761
## iter  40 value 150.925602
## iter  50 value 135.890675
## iter  60 value 127.564253
```

```
## iter  70 value 118.016448
## iter  80 value 105.359625
## iter  90 value 87.292017
## iter 100 value 73.964698
## final  value 73.964698
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 193.255856
## iter  20 value 169.566226
## iter  30 value 161.547617
## iter  40 value 150.922412
## iter  50 value 135.852402
## iter  60 value 128.170124
## iter  70 value 118.142897
## iter  80 value 106.923790
## iter  90 value 83.816943
## iter 100 value 67.162154
## final  value 67.162154
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 198.872461
## iter  20 value 160.365489
## iter  30 value 146.665649
## iter  40 value 130.908211
## iter  50 value 112.471216
## iter  60 value 100.596858
## iter  70 value 91.162305
## iter  80 value 81.850689
## iter  90 value 68.259332
## iter 100 value 58.319201
## final  value 58.319201
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 198.872502
## iter  20 value 160.366186
## iter  30 value 146.667909
## iter  40 value 130.912768
## iter  50 value 112.492289
## iter  60 value 100.749464
## iter  70 value 91.462314
## iter  80 value 81.991372
## iter  90 value 69.852755
## iter 100 value 61.827926
## final  value 61.827926
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 198.872461
## iter  20 value 160.365490
## iter  30 value 146.665652
## iter  40 value 130.908215
```

```
## iter  50 value 112.471237
## iter  60 value 100.597011
## iter  70 value 91.162599
## iter  80 value 81.851163
## iter  90 value 68.263030
## iter 100 value 58.319696
## final  value 58.319696
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 217.147130
## iter  20 value 185.174515
## iter  30 value 161.116208
## iter  40 value 138.699020
## iter  50 value 119.042502
## iter  60 value 104.898526
## iter  70 value 89.595790
## iter  80 value 75.027038
## iter  90 value 61.399459
## iter 100 value 48.599401
## final  value 48.599401
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 217.147199
## iter  20 value 185.175371
## iter  30 value 161.120607
## iter  40 value 138.727376
## iter  50 value 119.099421
## iter  60 value 105.179223
## iter  70 value 93.341589
## iter  80 value 82.252188
## iter  90 value 67.300520
## iter 100 value 55.318623
## final  value 55.318623
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 217.147130
## iter  20 value 185.174516
## iter  30 value 161.116213
## iter  40 value 138.699048
## iter  50 value 119.042558
## iter  60 value 104.898831
## iter  70 value 89.596302
## iter  80 value 75.027119
## iter  90 value 61.406127
## iter 100 value 48.588926
## final  value 48.588926
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 219.547244
## iter  20 value 180.679591
```

```
## iter  30 value 160.048801
## iter  40 value 145.015803
## iter  50 value 127.365485
## iter  60 value 115.098568
## iter  70 value 103.467705
## iter  80 value 86.933642
## iter  90 value 70.338412
## iter 100 value 47.047094
## final  value 47.047094
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 219.547294
## iter  20 value 180.680718
## iter  30 value 160.053386
## iter  40 value 145.026712
## iter  50 value 127.444920
## iter  60 value 115.256323
## iter  70 value 104.511383
## iter  80 value 88.454964
## iter  90 value 74.850636
## iter 100 value 57.084461
## final  value 57.084461
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 219.547244
## iter  20 value 180.679592
## iter  30 value 160.048806
## iter  40 value 145.015814
## iter  50 value 127.365564
## iter  60 value 115.098724
## iter  70 value 103.472091
## iter  80 value 86.938286
## iter  90 value 70.341737
## iter 100 value 48.774258
## final  value 48.774258
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 222.177237
## iter  20 value 183.695900
## iter  30 value 168.460614
## iter  40 value 141.775437
## iter  50 value 117.893750
## iter  60 value 104.493631
## iter  70 value 93.554224
## iter  80 value 83.244171
## iter  90 value 66.555929
## iter 100 value 51.587935
## final  value 51.587935
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
```

```
## iter  10 value 222.177259
## iter  20 value 183.696837
## iter  30 value 168.463578
## iter  40 value 141.796306
## iter  50 value 117.998250
## iter  60 value 104.482486
## iter  70 value 93.596729
## iter  80 value 84.332450
## iter  90 value 68.735382
## iter 100 value 55.682758
## final  value 55.682758
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 222.177237
## iter  20 value 183.695901
## iter  30 value 168.460617
## iter  40 value 141.775458
## iter  50 value 117.893855
## iter  60 value 104.493728
## iter  70 value 93.554245
## iter  80 value 83.246499
## iter  90 value 66.583788
## iter 100 value 51.631663
## final  value 51.631663
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 198.014483
## iter  20 value 163.243389
## iter  30 value 153.294231
## iter  40 value 139.007702
## iter  50 value 125.188116
## iter  60 value 111.291643
## iter  70 value 101.575636
## iter  80 value 89.721618
## iter  90 value 71.303326
## iter 100 value 57.827874
## final  value 57.827874
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 198.014521
## iter  20 value 163.244146
## iter  30 value 153.296581
## iter  40 value 139.018698
## iter  50 value 125.243505
## iter  60 value 111.453591
## iter  70 value 102.611496
## iter  80 value 91.699254
## iter  90 value 73.664960
## iter 100 value 60.654001
## final  value 60.654001
## stopped after 100 iterations
```

```
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 198.014483
## iter  20 value 163.243390
## iter  30 value 153.294233
## iter  40 value 139.007713
## iter  50 value 125.188171
## iter  60 value 111.291801
## iter  70 value 101.576192
## iter  80 value 89.720986
## iter  90 value 71.304857
## iter 100 value 57.832811
## final  value 57.832811
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 198.547283
## iter  20 value 161.413164
## iter  30 value 143.674534
## iter  40 value 129.666784
## iter  50 value 117.427029
## iter  60 value 106.920487
## iter  70 value 99.290137
## iter  80 value 88.577600
## iter  90 value 74.983563
## iter 100 value 61.483425
## final  value 61.483425
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 198.547307
## iter  20 value 161.413804
## iter  30 value 143.677791
## iter  40 value 129.677171
## iter  50 value 117.468268
## iter  60 value 107.082271
## iter  70 value 99.688864
## iter  80 value 90.828649
## iter  90 value 77.285758
## iter 100 value 66.376546
## final  value 66.376546
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 198.547283
## iter  20 value 161.413165
## iter  30 value 143.674538
## iter  40 value 129.666794
## iter  50 value 117.427071
## iter  60 value 106.920650
## iter  70 value 99.290536
## iter  80 value 88.579051
## iter  90 value 74.987010
## iter 100 value 61.489570
```

```
## final  value 61.489570
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 205.502591
## iter  20 value 162.893446
## iter  30 value 142.288745
## iter  40 value 119.405271
## iter  50 value 102.962989
## iter  60 value 89.505056
## iter  70 value 81.638490
## iter  80 value 68.457163
## iter  90 value 56.128761
## iter 100 value 44.743050
## final  value 44.743050
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 205.502628
## iter  20 value 162.894410
## iter  30 value 142.293061
## iter  40 value 119.427221
## iter  50 value 103.040804
## iter  60 value 89.722726
## iter  70 value 82.028644
## iter  80 value 69.584299
## iter  90 value 59.108737
## iter 100 value 50.117324
## final  value 50.117324
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 205.502591
## iter  20 value 162.893447
## iter  30 value 142.288750
## iter  40 value 119.405293
## iter  50 value 102.963067
## iter  60 value 89.505275
## iter  70 value 81.638882
## iter  80 value 68.458301
## iter  90 value 56.131296
## iter 100 value 44.765199
## final  value 44.765199
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 214.716518
## iter  20 value 176.689163
## iter  30 value 156.812372
## iter  40 value 134.129020
## iter  50 value 120.515334
## iter  60 value 107.875438
## iter  70 value 97.389940
## iter  80 value 86.543969
```

```
## iter  90 value 68.185241
## iter 100 value 56.977888
## final  value 56.977888
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 214.716542
## iter  20 value 176.689787
## iter  30 value 156.815977
## iter  40 value 134.145625
## iter  50 value 120.556271
## iter  60 value 107.881580
## iter  70 value 97.206975
## iter  80 value 87.038323
## iter  90 value 67.637656
## iter 100 value 60.037836
## final  value 60.037836
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 214.716518
## iter  20 value 176.689164
## iter  30 value 156.812375
## iter  40 value 134.129036
## iter  50 value 120.515374
## iter  60 value 107.875441
## iter  70 value 97.359029
## iter  80 value 86.340544
## iter  90 value 65.645124
## iter 100 value 53.276243
## final  value 53.276243
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 204.442352
## iter  20 value 174.006663
## iter  30 value 159.198296
## iter  40 value 141.764344
## iter  50 value 121.995490
## iter  60 value 111.577766
## iter  70 value 104.976861
## iter  80 value 95.322945
## iter  90 value 81.843666
## iter 100 value 68.167730
## final  value 68.167730
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 204.442381
## iter  20 value 174.007261
## iter  30 value 159.201944
## iter  40 value 141.776503
## iter  50 value 122.061278
## iter  60 value 111.737668
```

```
## iter  70 value 105.199697
## iter  80 value 96.191766
## iter  90 value 83.287598
## iter 100 value 72.926321
## final  value 72.926321
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 204.442352
## iter  20 value 174.006663
## iter  30 value 159.198300
## iter  40 value 141.764356
## iter  50 value 121.995556
## iter  60 value 111.577926
## iter  70 value 104.977091
## iter  80 value 95.323882
## iter  90 value 81.845490
## iter 100 value 68.173369
## final  value 68.173369
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 187.319409
## iter  20 value 163.880351
## iter  30 value 150.703799
## iter  40 value 139.581738
## iter  50 value 127.778797
## iter  60 value 118.184110
## iter  70 value 109.342966
## iter  80 value 97.692312
## iter  90 value 84.130133
## iter 100 value 65.987601
## final  value 65.987601
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 187.319431
## iter  20 value 163.880800
## iter  30 value 150.705832
## iter  40 value 139.587287
## iter  50 value 127.819713
## iter  60 value 118.309839
## iter  70 value 109.728933
## iter  80 value 99.155719
## iter  90 value 86.374408
## iter 100 value 70.772987
## final  value 70.772987
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 187.319409
## iter  20 value 163.880351
## iter  30 value 150.703801
## iter  40 value 139.581743
```

```
## iter  50 value 127.778838
## iter  60 value 118.184233
## iter  70 value 109.343337
## iter  80 value 97.713001
## iter  90 value 83.867982
## iter 100 value 65.856142
## final  value 65.856142
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 212.788280
## iter  20 value 182.976638
## iter  30 value 164.167229
## iter  40 value 147.014574
## iter  50 value 135.178012
## iter  60 value 120.988640
## iter  70 value 108.495043
## iter  80 value 90.289476
## iter  90 value 72.557102
## iter 100 value 56.311706
## final  value 56.311706
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 212.788305
## iter  20 value 182.977451
## iter  30 value 164.170405
## iter  40 value 147.029142
## iter  50 value 132.488925
## iter  60 value 117.277723
## iter  70 value 106.083225
## iter  80 value 88.992273
## iter  90 value 73.619353
## iter 100 value 60.268128
## final  value 60.268128
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 212.788280
## iter  20 value 182.976639
## iter  30 value 164.167232
## iter  40 value 147.014589
## iter  50 value 135.174191
## iter  60 value 120.981524
## iter  70 value 108.541740
## iter  80 value 90.391443
## iter  90 value 72.593948
## iter 100 value 54.766779
## final  value 54.766779
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 197.319282
## iter  20 value 168.691622
```

```
## iter  30 value 152.761202
## iter  40 value 140.436419
## iter  50 value 125.483449
## iter  60 value 114.923844
## iter  70 value 105.073702
## iter  80 value 89.895002
## iter  90 value 73.011094
## iter 100 value 60.394858
## final  value 60.394858
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 197.319324
## iter  20 value 168.692368
## iter  30 value 152.763976
## iter  40 value 140.450165
## iter  50 value 125.529048
## iter  60 value 114.459914
## iter  70 value 102.303077
## iter  80 value 88.680812
## iter  90 value 72.966423
## iter 100 value 62.306288
## final  value 62.306288
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 197.319282
## iter  20 value 168.691623
## iter  30 value 152.761205
## iter  40 value 140.436433
## iter  50 value 125.483495
## iter  60 value 114.924065
## iter  70 value 105.074083
## iter  80 value 89.895913
## iter  90 value 73.018049
## iter 100 value 60.404086
## final  value 60.404086
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 213.692134
## iter  20 value 179.413937
## iter  30 value 169.463624
## iter  40 value 158.306075
## iter  50 value 135.984806
## iter  60 value 124.725584
## iter  70 value 115.198492
## iter  80 value 102.572627
## iter  90 value 85.163882
## iter 100 value 71.843884
## final  value 71.843884
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
```

```
## iter  10 value 213.692161
## iter  20 value 179.414490
## iter  30 value 169.465657
## iter  40 value 158.314574
## iter  50 value 136.061408
## iter  60 value 124.910033
## iter  70 value 115.319235
## iter  80 value 103.722283
## iter  90 value 87.626579
## iter 100 value 75.330223
## final  value 75.330223
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 213.692134
## iter  20 value 179.413937
## iter  30 value 169.463626
## iter  40 value 158.306083
## iter  50 value 135.984882
## iter  60 value 124.725770
## iter  70 value 115.198612
## iter  80 value 102.573828
## iter  90 value 85.168921
## iter 100 value 71.809995
## final  value 71.809995
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 185.510958
## iter  20 value 153.779282
## iter  30 value 132.392051
## iter  40 value 114.750692
## iter  50 value 104.624872
## iter  60 value 94.442381
## iter  70 value 85.081105
## iter  80 value 74.726202
## iter  90 value 61.682851
## iter 100 value 47.204867
## final  value 47.204867
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 185.510993
## iter  20 value 153.779960
## iter  30 value 132.395847
## iter  40 value 114.766467
## iter  50 value 104.669043
## iter  60 value 94.598508
## iter  70 value 85.106075
## iter  80 value 75.257515
## iter  90 value 62.194104
## iter 100 value 49.924523
## final  value 49.924523
## stopped after 100 iterations
```

```
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 185.510958
## iter  20 value 153.779282
## iter  30 value 132.392055
## iter  40 value 114.750708
## iter  50 value 104.624916
## iter  60 value 94.442538
## iter  70 value 85.081235
## iter  80 value 74.726823
## iter  90 value 61.676661
## iter 100 value 47.030858
## final  value 47.030858
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 209.642571
## iter  20 value 179.759413
## iter  30 value 168.053332
## iter  40 value 153.694225
## iter  50 value 137.705260
## iter  60 value 129.847755
## iter  70 value 119.731961
## iter  80 value 109.025680
## iter  90 value 90.033765
## iter 100 value 74.829800
## final  value 74.829800
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 209.642592
## iter  20 value 179.759801
## iter  30 value 168.054778
## iter  40 value 153.704448
## iter  50 value 137.760421
## iter  60 value 130.512384
## iter  70 value 120.151533
## iter  80 value 109.370255
## iter  90 value 92.308439
## iter 100 value 77.805340
## final  value 77.805340
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 209.642571
## iter  20 value 179.759413
## iter  30 value 168.053334
## iter  40 value 153.694235
## iter  50 value 137.705315
## iter  60 value 129.847905
## iter  70 value 119.732265
## iter  80 value 109.026561
## iter  90 value 90.035983
## iter 100 value 74.833025
```

```
## final  value 74.833025
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 186.490921
## iter  20 value 162.144599
## iter  30 value 143.754711
## iter  40 value 126.825753
## iter  50 value 112.262999
## iter  60 value 102.669301
## iter  70 value 94.681078
## iter  80 value 86.211041
## iter  90 value 72.303845
## iter 100 value 61.631552
## final  value 61.631552
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 186.490950
## iter  20 value 162.145074
## iter  30 value 143.757599
## iter  40 value 126.841124
## iter  50 value 112.326556
## iter  60 value 102.826777
## iter  70 value 95.062857
## iter  80 value 86.895015
## iter  90 value 73.651762
## iter 100 value 64.104077
## final  value 64.104077
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 186.490921
## iter  20 value 162.144600
## iter  30 value 143.754714
## iter  40 value 126.825768
## iter  50 value 112.263062
## iter  60 value 102.669460
## iter  70 value 94.681464
## iter  80 value 86.211729
## iter  90 value 72.305184
## iter 100 value 61.652597
## final  value 61.652597
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 210.049053
## iter  20 value 171.988900
## iter  30 value 150.035879
## iter  40 value 131.267668
## iter  50 value 116.527996
## iter  60 value 103.612804
## iter  70 value 94.640350
## iter  80 value 77.347334
```

```
## iter  90 value 61.141791
## iter 100 value 45.768014
## final  value 45.768014
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 210.049072
## iter  20 value 171.989716
## iter  30 value 150.039507
## iter  40 value 131.285256
## iter  50 value 116.595890
## iter  60 value 103.812145
## iter  70 value 94.935155
## iter  80 value 78.314087
## iter  90 value 64.519665
## iter 100 value 53.527043
## final  value 53.527043
## stopped after 100 iterations
## # weights:  285 (224 variable)
## initial  value 270.385569
## iter  10 value 210.049053
## iter  20 value 171.988901
## iter  30 value 150.035883
## iter  40 value 131.267685
## iter  50 value 116.528064
## iter  60 value 103.613005
## iter  70 value 94.640631
## iter  80 value 77.348330
## iter  90 value 61.143720
## iter 100 value 45.770891
## final  value 45.770891
## 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 = "backward", 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        0.26263062
## 5       0.421579113       -0.2513299        0.15172993        0.04497442
##   `setting:prerelat` `setting:preclasf` `viewenc:prebody` `viewenc:prelet`
## 2          0.9158782        -0.20327566     -0.1732281310      -0.15753436
## 3         -0.6874554         0.77873656     -0.1003194116       0.22031347
## 4          0.2901539        -0.09297208      0.0008472878       0.10954771
## 5          0.1997050        -0.02496983     -0.4660637394       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       0.0001794691        0.072813219
## 3      -0.006339726        0.02589022      -0.0338563962       -0.005385365
## 4       0.029870629        0.01796772       0.0091114656        0.046634386
## 5       0.028841972        0.03215846      -0.0931836916        0.036711837
##   `prenumb:prerelat` `prenumb:preclasf` `prerelat:preclasf`
## 2       0.0347272942        0.034704048          -0.2159680
## 3      -0.0023133262        0.006896269          -0.1384417
## 4      -0.0002585178       -0.019663447          -0.1506193
## 5       0.0255919064       -0.001445600          -0.2108050
##
## Std. Errors:
##   (Intercept)       age    viewcat     setting    viewenc   prebody    prelet
## 2  0.01769491 0.2281321 0.04463296 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 imporatnce variance for multiclass classification

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