Group5 Code

Bingyu Sun

4/4/2019

Tidy data

```
apple_with_vpp = read_csv("./data/AppleStore.csv") %>%
  janitor::clean_names() %>%
  dplyr::select(-c(x1, id, track_name, currency, ver)) %>%
  mutate(size_bytes = round(size_bytes * 1e-6)) %>%
  rename(size_megabytes = size_bytes) %>%
  filter(rating_count_tot != 0) %>% #Remove apps with no user rating
  mutate(prime_genre = as.integer(ifelse(prime_genre == "Games", 1, 0))) %>%
  dplyr::select(-rating_count_tot, -rating_count_ver) #with vpp_lic
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   id = col_double(),
##   track_name = col_character(),
##   size_bytes = col_double(),
##   currency = col_character(),
##   price = col_double(),
##   rating_count_tot = col_double(),
##   rating_count_ver = col_double(),
##   user_rating = col_double(),
##   user_rating_ver = col_double(),
##   ver = col_character(),
##   cont_rating = col_character(),
##   prime_genre = col_character(),
##   sup_devices.num = col_double(),
##   ipadSc_urls.num = col_double(),
##   lang.num = col_double(),
##   vpp_lic = col_double()
## )
```

```
apple = read_csv("./data/AppleStore.csv") %>%
  janitor::clean_names() %>%
  dplyr::select(-c(x1, id, track_name, currency, ver)) %>%
  mutate(size_bytes = round(size_bytes * 1e-6)) %>%
  rename(size_megabytes = size_bytes) %>%
  filter(rating_count_tot != 0) %>% #Remove apps with no user rating
  mutate(prime_genre = as.integer(ifelse(prime_genre == "Games", 1, 0))) %>%
  dplyr::select(-rating_count_tot, -rating_count_ver, -vpp_lic) #vpp_lic has nearzero variance
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
```

```
##   id = col_double(),
##   track_name = col_character(),
##   size_bytes = col_double(),
##   currency = col_character(),
##   price = col_double(),
##   rating_count_tot = col_double(),
##   rating_count_ver = col_double(),
##   user_rating = col_double(),
##   user_rating_ver = col_double(),
##   ver = col_character(),
##   cont_rating = col_character(),
##   prime_genre = col_character(),
##   sup_devices.num = col_double(),
##   ipadSc_urls.num = col_double(),
##   lang.num = col_double(),
##   vpp_lic = col_double()
## )
```

```r
skimr::skim(apple)
```

```
## Skim summary statistics
##  n obs: 6268
##  n variables: 9
##
## -- Variable type:character -----------------------------------------------------------------
##     variable missing complete    n min max empty n_unique
##  cont_rating       0     6268 6268   2   3     0        4
##
## -- Variable type:integer -------------------------------------------------------------------
##     variable missing complete    n mean  sd p0 p25 p50 p75 p100     hist
##  prime_genre       0     6268 6268 0.54 0.5  0   0   1   1    1
##
## -- Variable type:numeric -------------------------------------------------------------------
##          variable missing complete    n   mean     sd p0  p25   p50    p75
##  ipad_sc_urls_num       0     6268 6268   3.87   1.88  0  4     5      5
##          lang_num       0     6268 6268   5.89   8.2   0  1     1      9
##             price       0     6268 6268   1.82   6.13  0  0     0      2.99
##    size_megabytes       0     6268 6268 205.74 352.63  1 52   102    188.25
##   sup_devices_num       0     6268 6268  37.26   3.91  9 37    37     38
##       user_rating       0     6268 6268   4.05   0.73  1  4     4.5    4.5
##   user_rating_ver       0     6268 6268   3.74   1.4   0  3.5   4.5    4.5
##     p100     hist
##     5
##    75
##   299.99
##  4026
##    47
##     5
##     5
```

```r
#matrix of predictors
x = model.matrix(user_rating~., apple)[,-1]
y = apple$user_rating
```

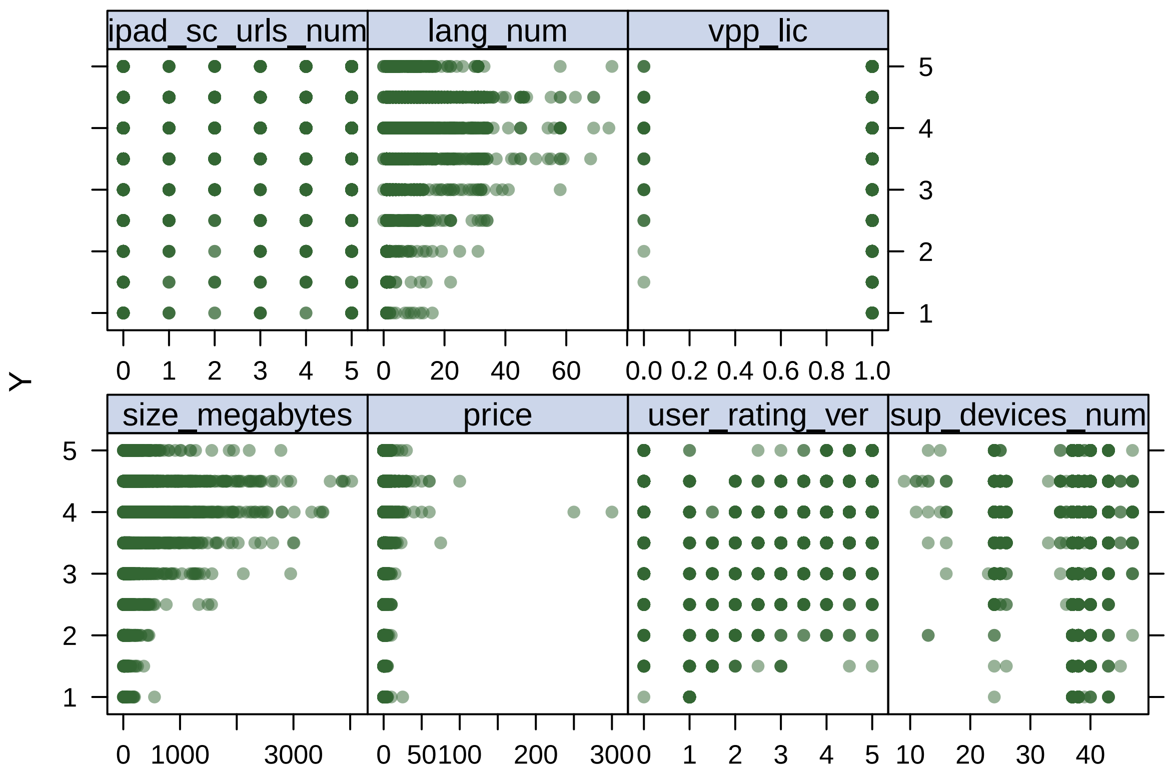
EDA

```r
#boxplots for categorical variables
apple %>%
  mutate(cont_rating = forcats::fct_reorder(cont_rating, user_rating)) %>%
  ggplot(aes(x = cont_rating, y = user_rating)) +
  geom_boxplot()

apple %>%
  mutate(prime_genre = as.factor(prime_genre)) %>%
  mutate(prime_genre = forcats::fct_reorder(prime_genre, user_rating)) %>%
  ggplot(aes(x = prime_genre, y = user_rating)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

#histograms for response
apple %>%
  ggplot(aes(x = user_rating)) +
  geom_histogram()

#Correlation (no cont_rating, prime_genre)
cor_matrix = model.matrix(user_rating ~., data = apple)[,-1]
corrplot::corrplot(cor(cor_matrix))

#Scatterplots
apple %>%
  ggplot(aes(x = rating_count_tot, y = user_rating)) +
  geom_point(alpha = .5) +
  stat_smooth(method = "lm")

apple %>%
  ggplot(aes(x = rating_count_ver, y = user_rating)) +
  geom_point(alpha = .5) +
  stat_smooth(method = "lm")

apple %>%
  ggplot(aes(x = size_megabytes, y = user_rating)) +
  geom_point(alpha = .5) +
  stat_smooth(method = "lm")

apple %>%
  ggplot(aes(x = price, y = user_rating)) +
  geom_point(alpha = .5) +
  stat_smooth(method = "lm")

apple_with_vpp %>%
  select(-cont_rating, -prime_genre) %>%
  select(user_rating, user_rating_ver, ipad_sc_urls_num, vpp_lic) %>%
  gather(key = variables, value = x, user_rating_ver:vpp_lic) %>%
  ggplot(aes(x = x, y = user_rating)) +
  geom_point(alpha = .5) +
  stat_smooth(method = "lm") +
  facet_grid(~variables)
```

```r
apple %>%
  select(-cont_rating, -prime_genre) %>%
  select(user_rating, sup_devices_num, lang_num) %>%
  gather(key = variables, value = x, sup_devices_num:lang_num) %>%
  ggplot(aes(x = x, y = user_rating)) +
  geom_point(alpha = .5) +
  stat_smooth(method = "lm") +
  facet_grid(~variables)
```

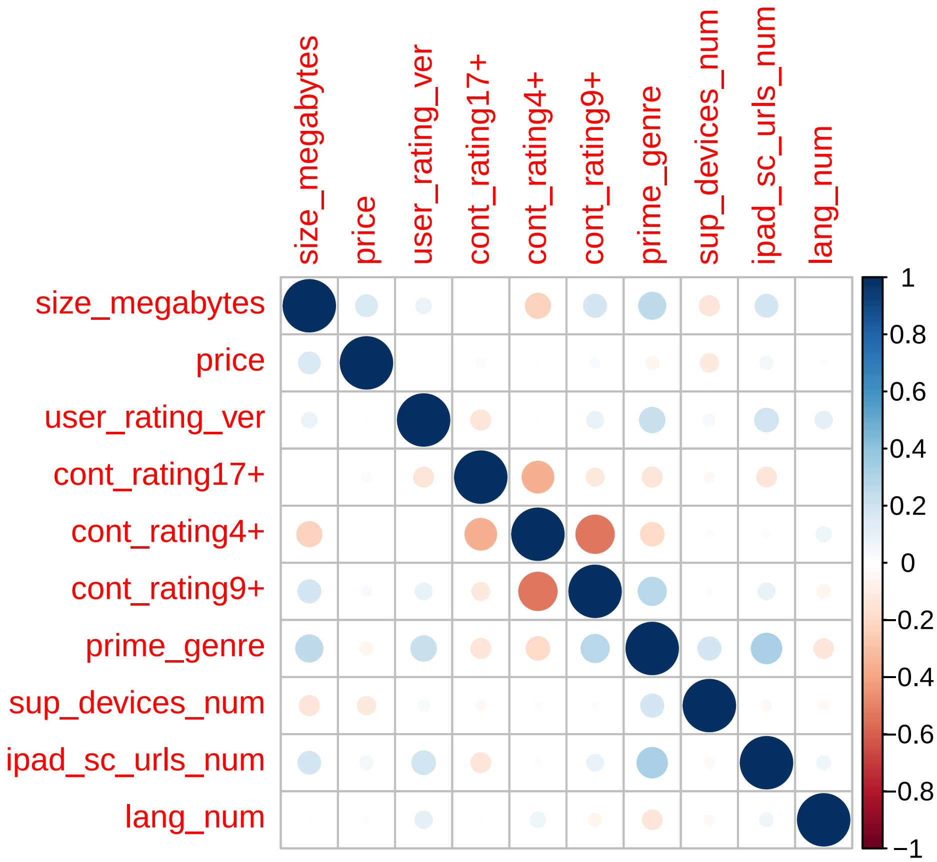
Feature plot

```r
theme1 <- trellis.par.get()
theme1$plot.symbol$col <- rgb(.2, .4, .2, .5)
theme1$plot.symbol$pch <- 16
theme1$plot.line$col <- rgb(.8, .1, .1, 1)
theme1$plot.line$lwd <- 2
theme1$strip.background$col <- rgb(.0, .2, .6, .2)
trellis.par.set(theme1)
featurePlot(apple_with_vpp[, -c(3,5,6)], apple$user_rating, plot = "scatter", labels = c("","Y"),
            type = c("p"), layout = c(4,2)) # plot numerical values
```



Correlation (no content rating and primary genre)

```r
cor_matrix = model.matrix(user_rating ~., data = apple)[,-1]
corrplot::corrplot(cor(cor_matrix))
```



Check linear dependency of numerical predictors (no problematic predictors)

```
nzv <- nearZeroVar(apple_with_vpp) # nzv: near zero variance
apple_2 <- apple_with_vpp[, -nzv]
combInfo <- findLinearCombos(apple_2[,-c(5)]) # on numerical values
combInfo # no linear dependency problem
```

```
## $linearCombos
## list()
##
## $remove
## NULL
```

```
names(apple_2) # remove vpp_lic; 8 predictors + 1 response
```

```
## [1] "size_megabytes"   "price"            "user_rating"
## [4] "user_rating_ver"  "cont_rating"      "prime_genre"
## [7] "sup_devices_num"  "ipad_sc_urls_num" "lang_num"
```

Box-cox

```
mult.fit1 <- lm(user_rating ~ size_megabytes + price + user_rating_ver + cont_rating + prime_genre + sup
boxcox(mult.fit1)
```

Split to train/test sets

```
set.seed(1234)
#Split data to traning and testing
trRows = createDataPartition(apple$user_rating,
                             p = .75,
                             list = FALSE)
train_data = apple[trRows,]
test_data = apple[-trRows,]
#in matrix form
x_train = model.matrix(user_rating~., train_data)[,-1]
y_train = train_data$user_rating
x_test = model.matrix(user_rating~., test_data)[,-1]
y_test = test_data$user_rating

ctrl1 = trainControl(method = "repeatedcv", number = 10, repeats = 5)
```

Fit linear regression & 10-fold repeatedCV (5 times)

```
set.seed(1234)
lm.fit = train(user_rating~.,
               data = train_data,
               method = "lm",
               trControl = ctrl1)
lm.fit #RMSE 0.6289451
```

```
## Linear Regression
##
## 4702 samples
##    8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 4233, 4231, 4231, 4231, 4232, 4232, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.6289451  0.2547178  0.4581656
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
summary(lm.fit)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7370 -0.2640  0.1372  0.3236  2.0354
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)       2.963e+00  9.958e-02  29.759  < 2e-16 ***
## size_megabytes   -5.369e-05  2.998e-05  -1.791  0.07338 .
## price             4.426e-03  1.845e-03   2.399  0.01650 *
## user_rating_ver   2.367e-01  6.893e-03  34.335  < 2e-16 ***
## `cont_rating17+`  3.643e-02  4.157e-02   0.876  0.38081
## `cont_rating4+`  -5.827e-02  2.610e-02  -2.232  0.02564 *
## `cont_rating9+`  -7.236e-03  3.407e-02  -0.212  0.83183
## prime_genre       1.352e-01  2.217e-02   6.099 1.16e-09 ***
## sup_devices_num   1.577e-03  2.455e-03   0.643  0.52056
## ipad_sc_urls_num  2.144e-02  5.310e-03   4.037 5.50e-05 ***
## lang_num          4.043e-03  1.115e-03   3.627  0.00029 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6282 on 4691 degrees of freedom
## Multiple R-squared:  0.2566, Adjusted R-squared:  0.2551
## F-statistic:   162 on 10 and 4691 DF,  p-value: < 2.2e-16
```

Ridge, Lasso, and elastic net

Ridge

```r
# fit the ridge regression (alpha = 0) with a sequence of lambdas
ridge.mod = glmnet(x_train, y_train, alpha = 0, lambda = exp(seq(-8, 2, length = 100)))
dim(coef(ridge.mod))

#CV: get optimal lambda
set.seed(1234)
cv.ridge = cv.glmnet(x_train, y_train,
                      alpha = 0,
                      lambda = exp(seq(-8, 2, length = 100)),
                      type.measure = "mse")
plot(cv.ridge)

cv.ridge$lambda.min
```

Lasso (variable selection)

```r
cv.lasso = cv.glmnet(x_train, y_train,
                     alpha = 1,
                     lambda = exp(seq(-6, 2, length = 100)))
plot(cv.lasso)
cv.lasso$lambda.min #optimal lambda

predict(cv.lasso, s = "lambda.min", type = "coefficients")
```

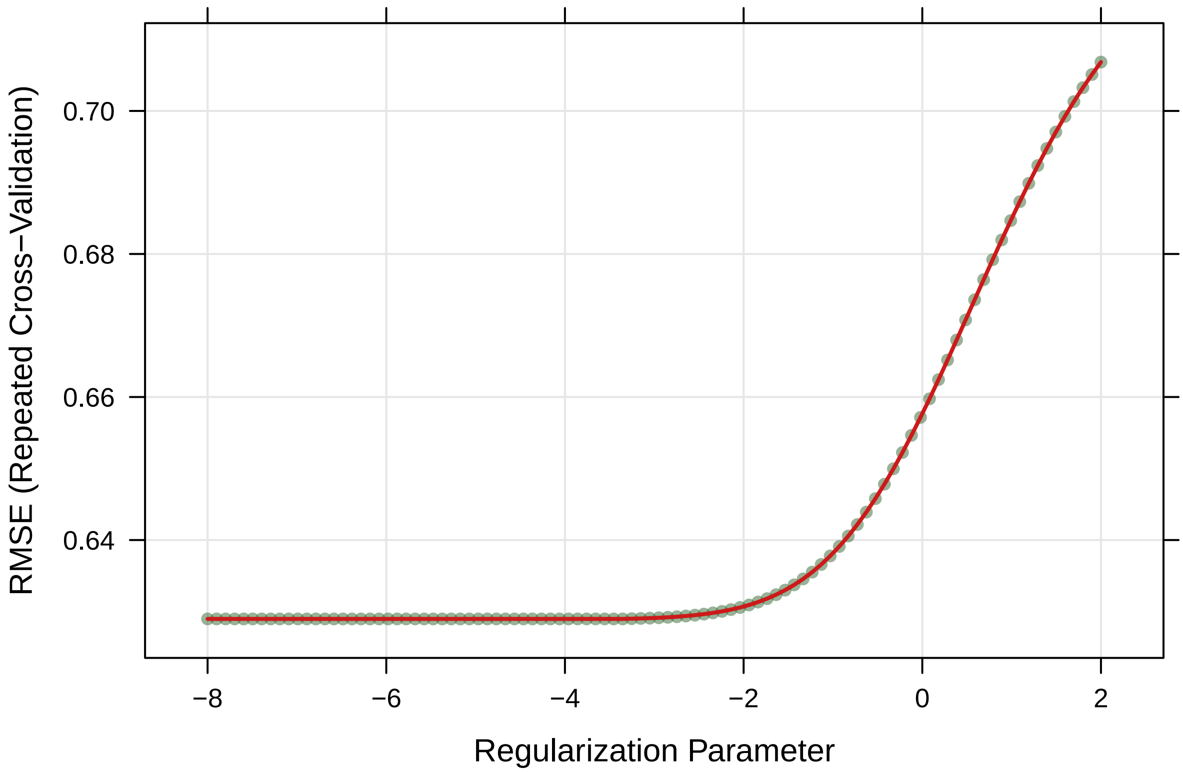
Use Caret (ridge and lasso)

```r
set.seed(1234)
ridge.fit <- train(x_train, y_train, method = "glmnet",
                   tuneGrid = expand.grid(alpha = 0,
                                          lambda = exp(seq(-8, 2, length = 100))),
                   trControl = ctrl1)
```

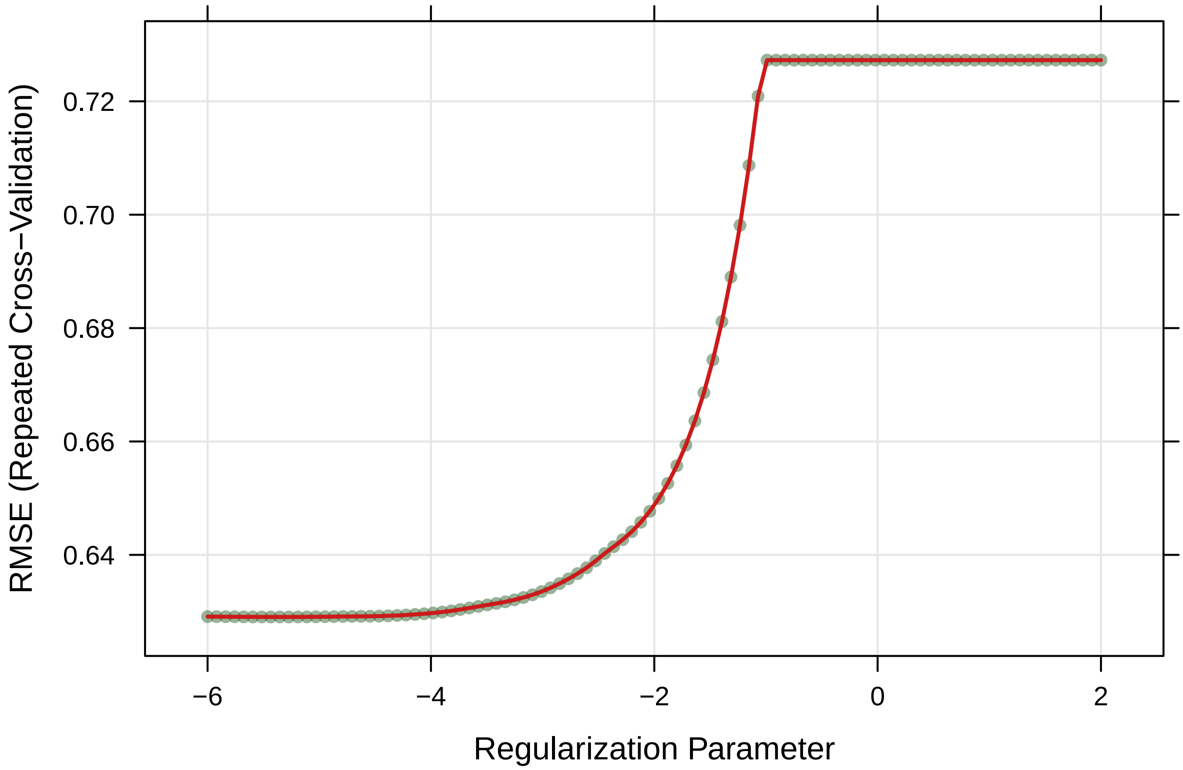
```r
lasso.fit <- train(x_train, y_train, method = "glmnet",
                   tuneGrid = expand.grid(alpha = 1,
                                          lambda = exp(seq(-6, 2, length = 100))),
                   trControl = ctrl1)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

```r
plot(ridge.fit, xTrans = function(x) log(x))
```




```r
plot(lasso.fit, xTrans = function(x) log(x))
```




```r
ridge.fit$bestTune #0.03
```

```
##    alpha     lambda
## 46     0 0.03160167
```

```r
lasso.fit$bestTune #0.005
```

```
##   alpha      lambda
## 9     1 0.004731394
```

```r
coef(lasso.fit$finalModel, lasso.fit$bestTune$lambda) #get covariates
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                              1
## (Intercept)       3.013686e+00
## size_megabytes   -2.717265e-05
## price             3.049752e-03
## user_rating_ver   2.341925e-01
## cont_rating17+    1.736256e-02
## cont_rating4+    -4.725665e-02
## cont_rating9+     .
## prime_genre       1.231547e-01
## sup_devices_num   7.099613e-04
## ipad_sc_urls_num  1.943850e-02
## lang_num          3.390353e-03
```

Elastic net

```r
set.seed(1234)
enet.fit <- train(x_train, y_train,
                  method = "glmnet",
```

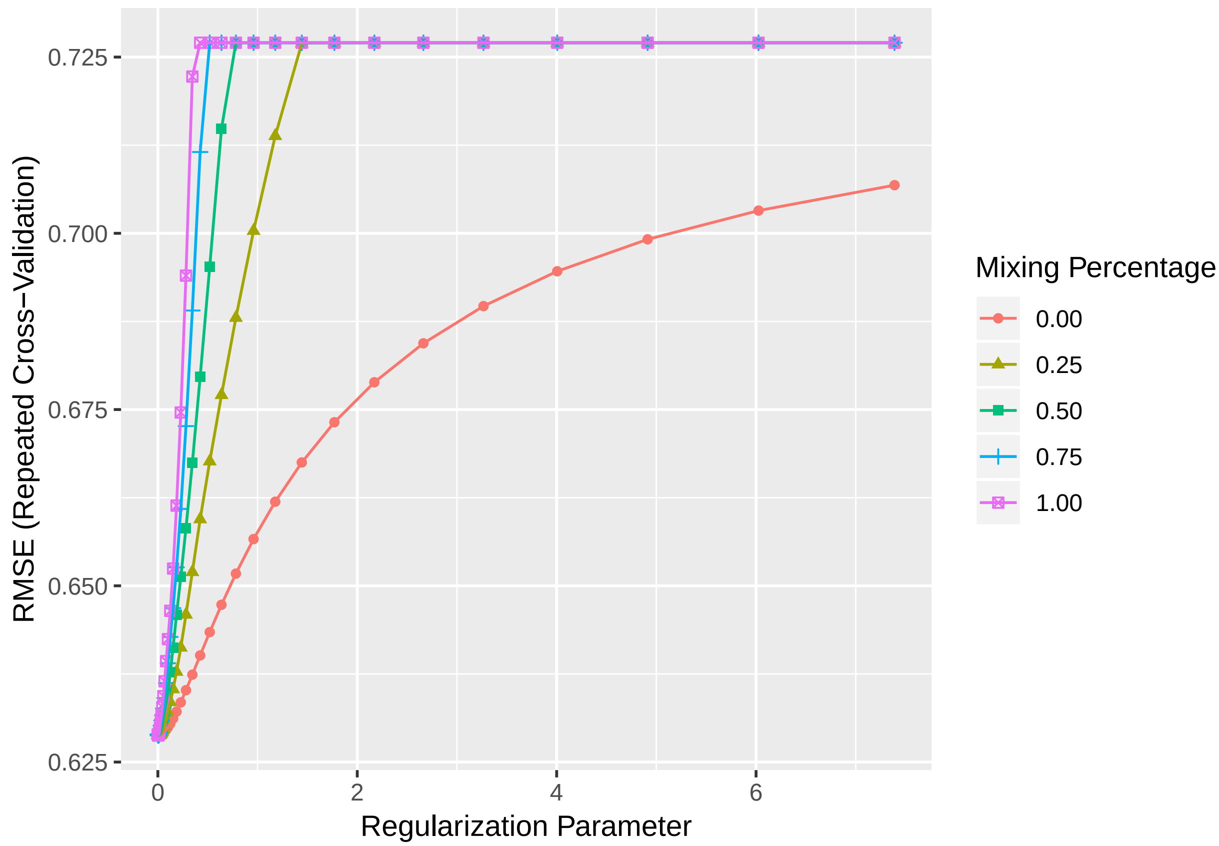
```r
                        tuneGrid = expand.grid(alpha = seq(0, 1, length = 5),
                                               lambda = exp(seq(-8, 2, length = 50))),
                        trControl = ctrl1)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info =
## trainInfo, : There were missing values in resampled performance measures.
```

```r
enet.fit$bestTune #0.004
```

```
##     alpha      lambda
## 213     1 0.003883492
```

```r
ggplot(enet.fit)
```



PCR and PLS

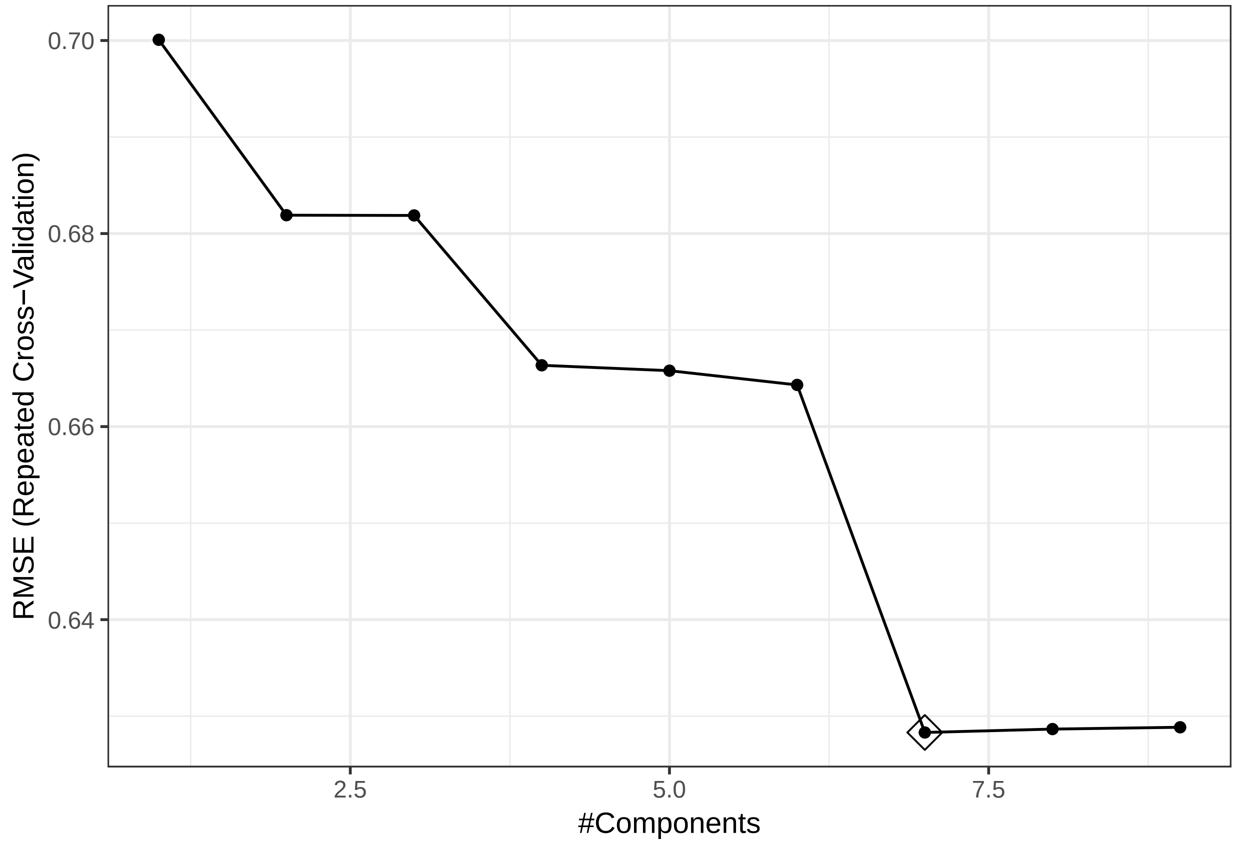
PCR

```r
set.seed(1234)
pcr.fit <- train(x_train, y_train,
                 method = "pcr",
                 tuneLength = 10,
                 trControl = ctrl1,
                 scale = TRUE)

pred.pcr <- predict(pcr.fit$finalModel, newdata = x_test,
                    ncomp = pcr.fit$bestTune[[1]], type = "response") #7
```

```
mean((pred.pcr - y_test)^2) #0.407
```

```
## [1] 0.4068258
```

```
ggplot(pcr.fit, highlight = TRUE) + theme_bw()
```

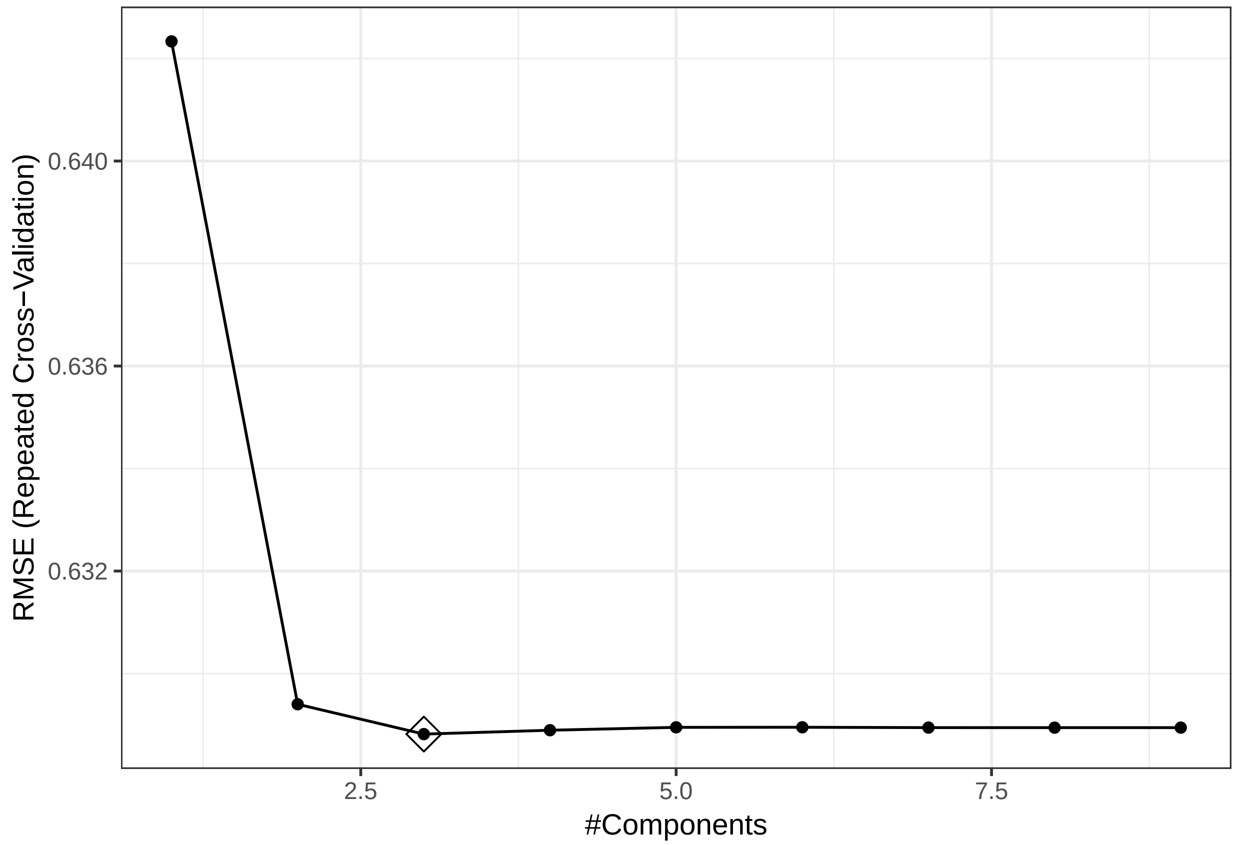


PLS

```
set.seed(1234)
pls.fit <- train(x_train, y_train,
                  method = "pls",
                  tuneLength = 10,
                  trControl = ctrl1,
                  scale = TRUE)

pred.pls <- predict(pls.fit$finalModel, newdata = x_test,
                     ncomp = pls.fit$bestTune[[1]]) # 3
mean((pred.pls - y_test)^2) # 0.407
```

```
## [1] 0.4065176
```

```
ggplot(pls.fit, highlight = TRUE) + theme_bw()
```



Non-linear

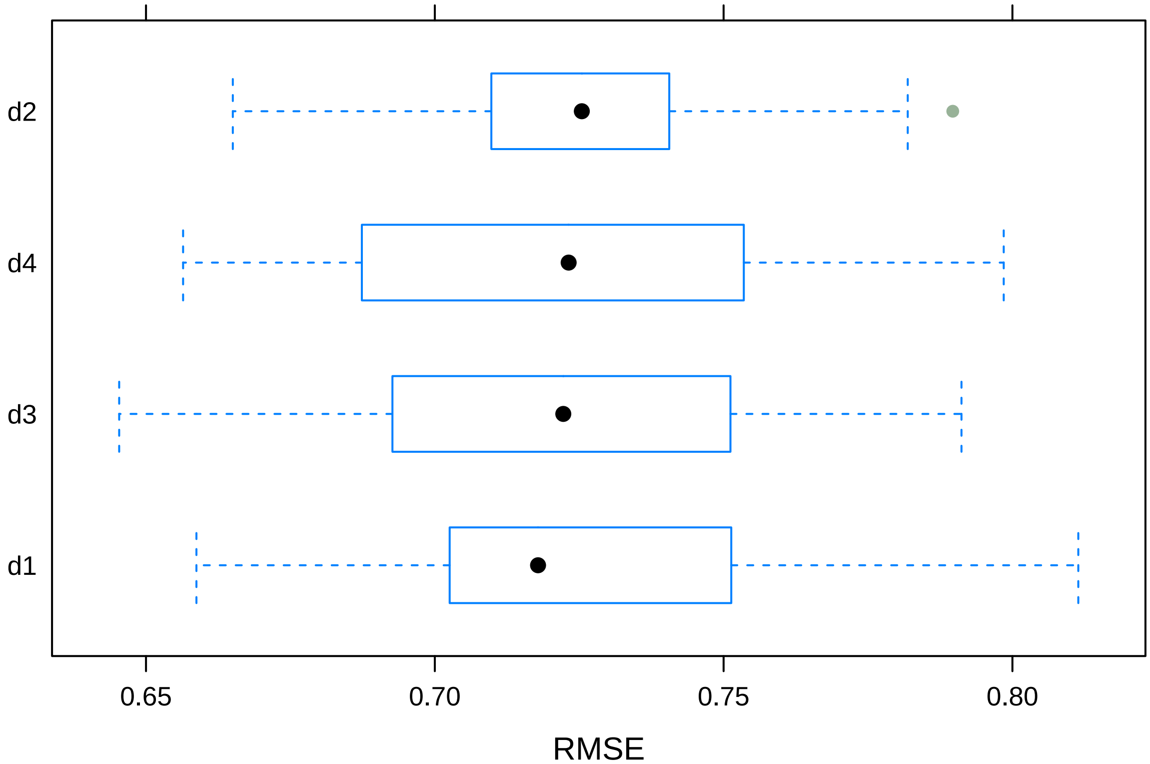
Polynomials

CV to compare models up to d = 4 and make plot

add higher order on size_megabytes

```
set.seed(1234)
lmFit1 <- train(user_rating ~ size_megabytes,
                data = train_data,
                method = "lm",
                trControl = ctrl1)
lmFit2 <- train(user_rating ~ poly(size_megabytes,2),
                data = train_data,
                method = "lm",
                trControl = ctrl1)
lmFit3 <- train(user_rating ~ poly(size_megabytes,3),
                data = train_data,
                method = "lm",
                trControl = ctrl1)
lmFit4 <- train(user_rating ~ poly(size_megabytes,4),
                data = train_data,
                method = "lm",
                trControl = ctrl1)

resamp <- resamples(list(d1 = lmFit1, d2 = lmFit2, d3 = lmFit3, d4 = lmFit4))
```

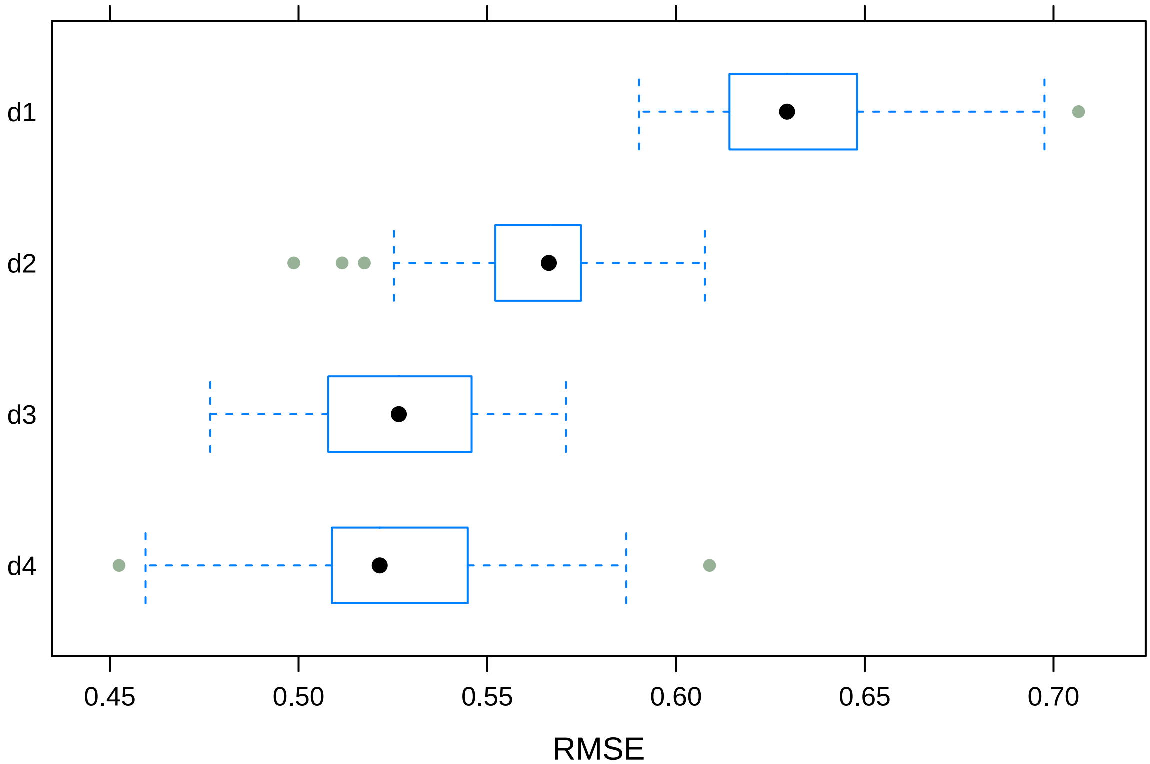
```r
#summary(resamp) # RMSE

bwplot(resamp, metric = "RMSE")
```



comment: d = 1

add higher order on user_rating_ver

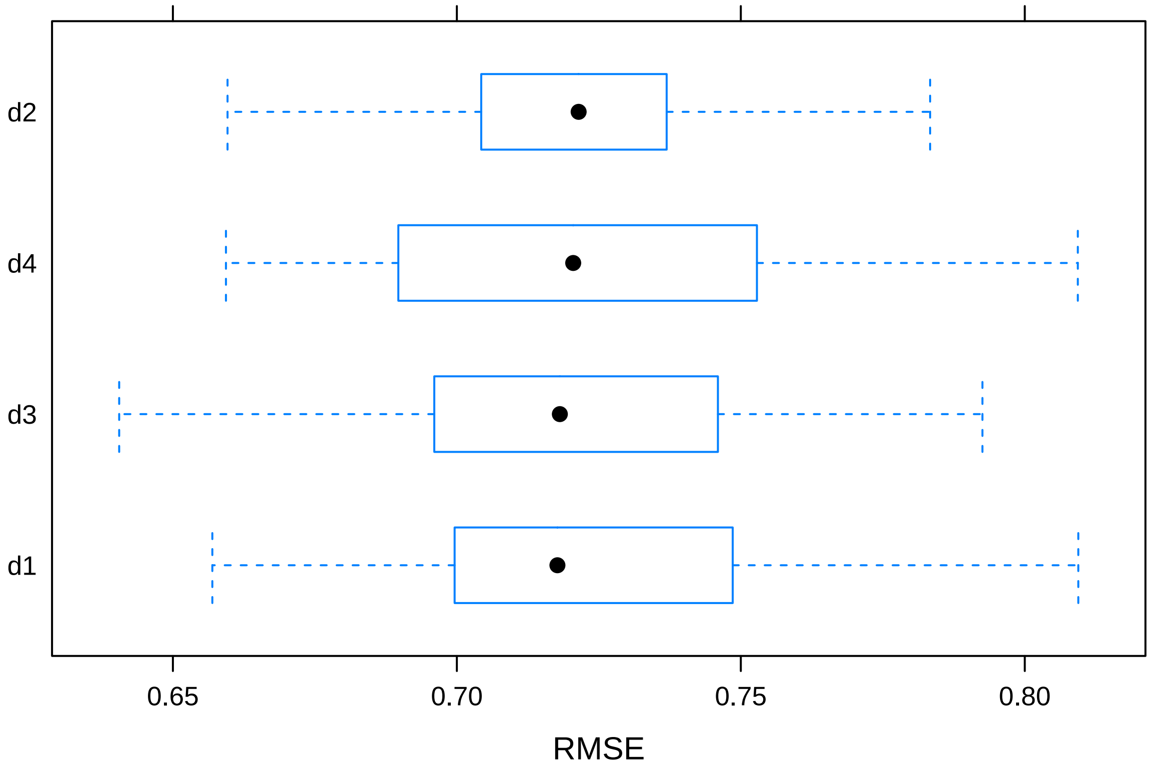
```r
set.seed(1234)
lmFit1 <- train(user_rating ~ user_rating_ver,
                data = train_data,
                method = "lm",
                trControl = ctrl1)
lmFit2 <- train(user_rating ~ poly(user_rating_ver,2),
                data = train_data,
                method = "lm",
                trControl = ctrl1)
lmFit3 <- train(user_rating ~ poly(user_rating_ver,3),
                data = train_data,
                method = "lm",
                trControl = ctrl1)
lmFit4 <- train(user_rating ~ poly(user_rating_ver,4),
                data = train_data,
                method = "lm",
                trControl = ctrl1)

resamp <- resamples(list(d1 = lmFit1, d2 = lmFit2, d3 = lmFit3, d4 = lmFit4))
#summary(resamp) # MSE

bwplot(resamp, metric = "RMSE")
```



Comment: d = 3 on user_rating_ver

add higher order on lang_num

```r
set.seed(1234)
lmFit1 <- train(user_rating ~ lang_num,
                data = train_data,
                method = "lm",
                trControl = ctrl1)
lmFit2 <- train(user_rating ~ poly(lang_num,2),
                data = train_data,
                method = "lm",
                trControl = ctrl1)
lmFit3 <- train(user_rating ~ poly(lang_num,3),
                data = train_data,
                method = "lm",
                trControl = ctrl1)
lmFit4 <- train(user_rating ~ poly(lang_num,4),
                data = train_data,
                method = "lm",
                trControl = ctrl1)

resamp <- resamples(list(d1 = lmFit1, d2 = lmFit2, d3 = lmFit3, d4 = lmFit4))
#summary(resamp) # MSE

bwplot(resamp, metric = "RMSE")
```



Conclusion: add polynomial component of lang_num and size_megabytes does not make too much difference

Comment: it does not really improve too much; so better keep d = 1

check anova

```
fit1 <- lm(user_rating~size_megabytes, data = train_data)  # y ~ X
fit2 <- lm(user_rating~poly(size_megabytes,2), data = train_data) # y ~ X + X^2
fit3 <- lm(user_rating~poly(size_megabytes,3), data = train_data) # y ~ X + X^2 + X^3
fit4 <- lm(user_rating~poly(size_megabytes,4), data = train_data) # y ~ X + X^2 + X^3 + X^4
anova(fit1, fit2, fit3, fit4)
```

```
## Analysis of Variance Table
##
## Model 1: user_rating ~ size_megabytes
## Model 2: user_rating ~ poly(size_megabytes, 2)
## Model 3: user_rating ~ poly(size_megabytes, 3)
## Model 4: user_rating ~ poly(size_megabytes, 4)
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1   4700 2480.8
## 2   4699 2473.9  1    6.9131 13.251 0.0002754 ***
## 3   4698 2461.3  1   12.6662 24.278 8.623e-07 ***
## 4   4697 2450.5  1   10.8132 20.727 5.431e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
fit1 <- lm(user_rating~user_rating_ver, data = train_data)  # y ~ X
fit2 <- lm(user_rating~poly(user_rating_ver,2), data = train_data) # y ~ X + X^2
fit3 <- lm(user_rating~poly(user_rating_ver,3), data = train_data) # y ~ X + X^2 + X^3
fit4 <- lm(user_rating~poly(user_rating_ver,4), data = train_data) # y ~ X + X^2 + X^3 + X^4
anova(fit1, fit2, fit3, fit4)
```

```
## Analysis of Variance Table
##
## Model 1: user_rating ~ user_rating_ver
## Model 2: user_rating ~ poly(user_rating_ver, 2)
## Model 3: user_rating ~ poly(user_rating_ver, 3)
## Model 4: user_rating ~ poly(user_rating_ver, 4)
##   Res.Df    RSS Df Sum of Sq       F    Pr(>F)
## 1   4700 1893.5
## 2   4699 1489.1  1    404.39 1460.69 < 2.2e-16 ***
## 3   4698 1304.5  1    184.61  666.82 < 2.2e-16 ***
## 4   4697 1300.4  1      4.14   14.95 0.0001119 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
fit1 <- lm(user_rating~lang_num, data = train_data)  # y ~ X
fit2 <- lm(user_rating~poly(lang_num,2), data = train_data) # y ~ X + X^2
fit3 <- lm(user_rating~poly(lang_num,3), data = train_data) # y ~ X + X^2 + X^3
fit4 <- lm(user_rating~poly(lang_num,4), data = train_data)
anova(fit1, fit2, fit3, fit4)
```

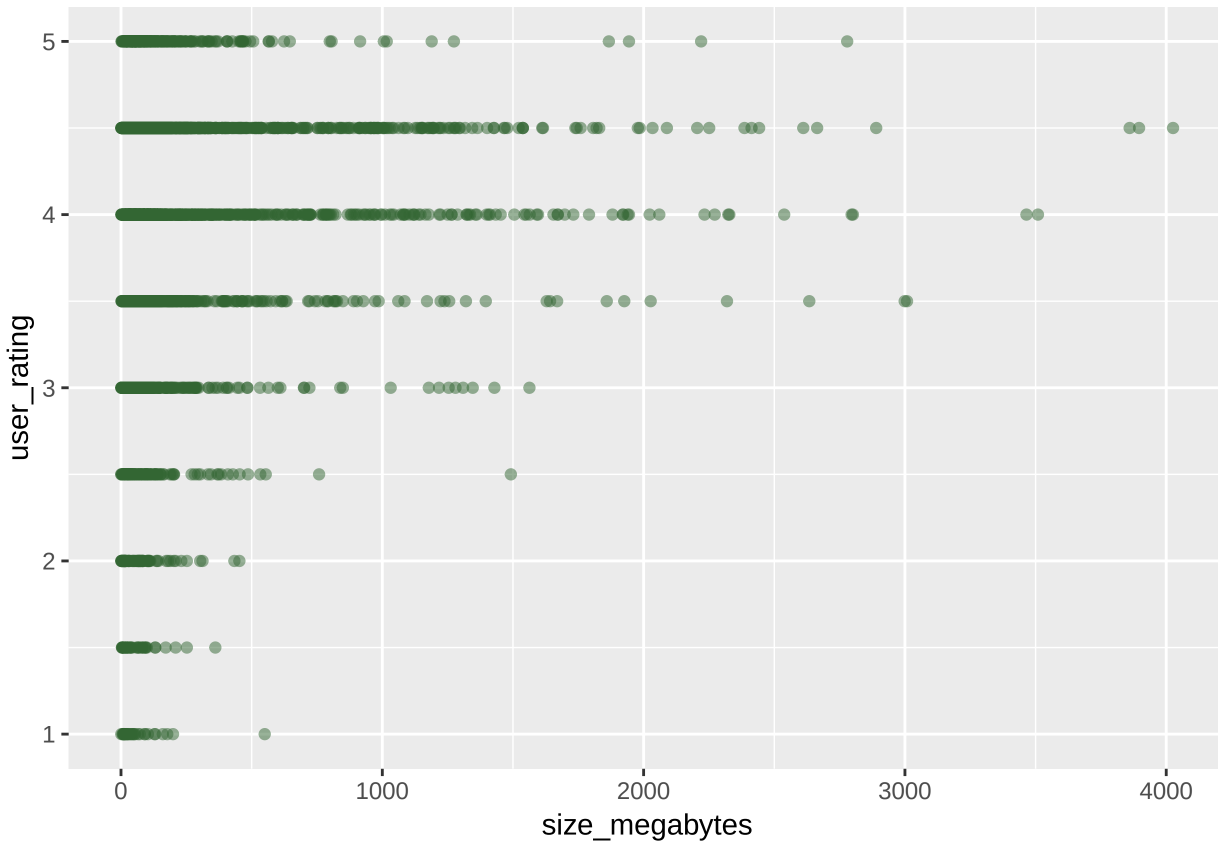
```
## Analysis of Variance Table
##
## Model 1: user_rating ~ lang_num
## Model 2: user_rating ~ poly(lang_num, 2)
## Model 3: user_rating ~ poly(lang_num, 3)
## Model 4: user_rating ~ poly(lang_num, 4)
##   Res.Df    RSS Df Sum of Sq       F    Pr(>F)
## 1   4700 2473.9
## 2   4699 2455.8  1   18.1218 34.8627 3.787e-09 ***
## 3   4698 2442.2  1   13.5944 26.1529 3.280e-07 ***
## 4   4697 2441.5  1    0.6964  1.3397    0.2472
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Comment: ANOVA result suggest adding d = 3 to lang_num and d = 4 on size_megabytes How to decide on this?

Smoothing splines

fit smoothing spline on size_megabytes

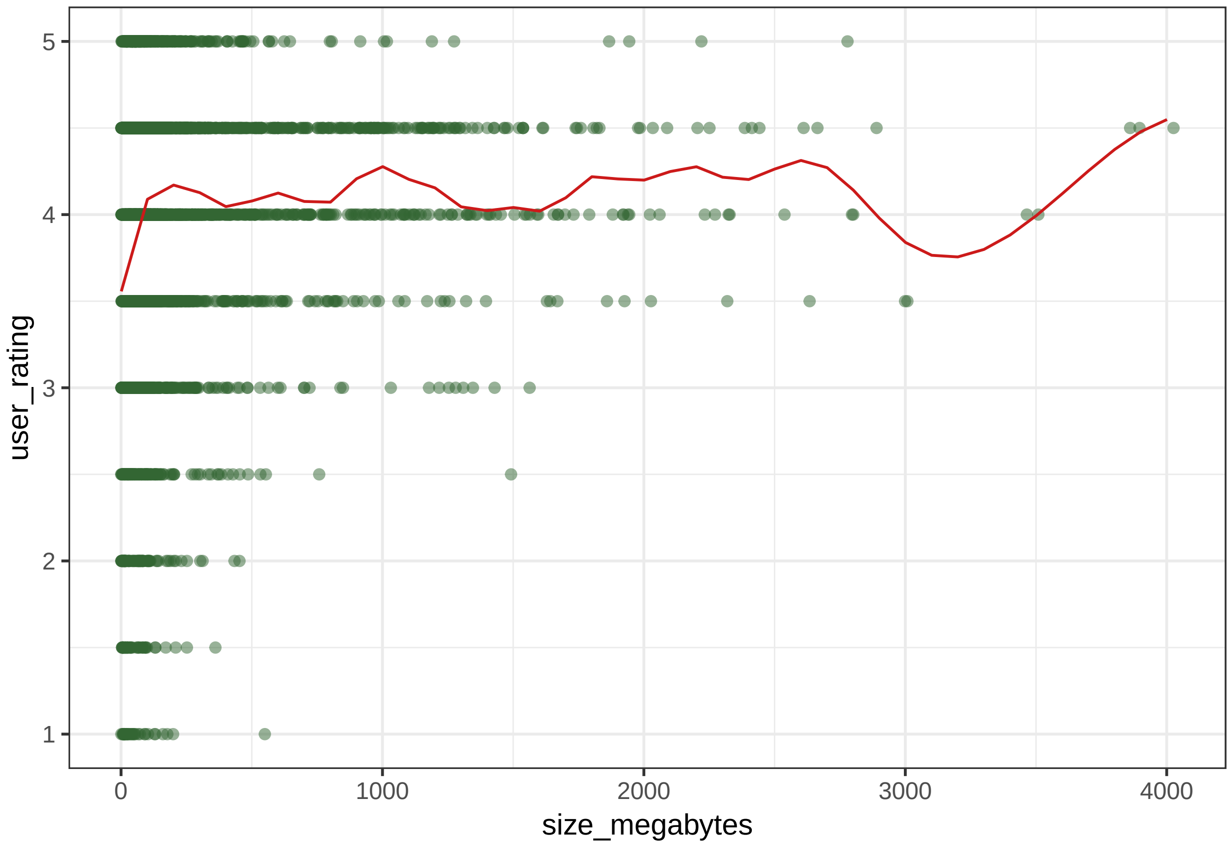
```r
p <- ggplot(data = train_data, aes(x = size_megabytes, y = user_rating)) +
     geom_point(color = rgb(.2, .4, .2, .5))
p
```




```r
fit.ss <- smooth.spline(train_data$size_megabytes, # predictor (univariate)
                        train_data$user_rating)  # response
fit.ss$df # 14.5
```

```
## [1] 23.22019
```

```r
# look at the range
sizelims <- range(apple$size_megabytes)

# create a sequence of observations pgg45
size.grid <- seq(from = sizelims[1],to = sizelims[2], 100)

pred.ss <- predict(fit.ss, x = size.grid) # specify x;
# but we did not calculate CI in this function
pred.ss.df <- data.frame(pred = pred.ss$y,
                         size = size.grid)

p +
geom_line(aes(x = size, y = pred), data = pred.ss.df,
          color = rgb(.8, .1, .1, 1)) + theme_bw()
```



fit smoothing spline to lang_num

```
fit.ss <- smooth.spline(train_data$lang_num, # predictor (univariate)
                        train_data$user_rating)  # response
fit.ss$df
```

```
## [1] 4.865362
```

```
# look at the range
langlims <- range(train_data$lang_num)

# create a sequence of observations pgg45
lang.grid <- seq(from = langlims[1],to = langlims[2])
lang.grid
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
## [24] 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
## [47] 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
## [70] 69 70 71 72 73 74 75
```

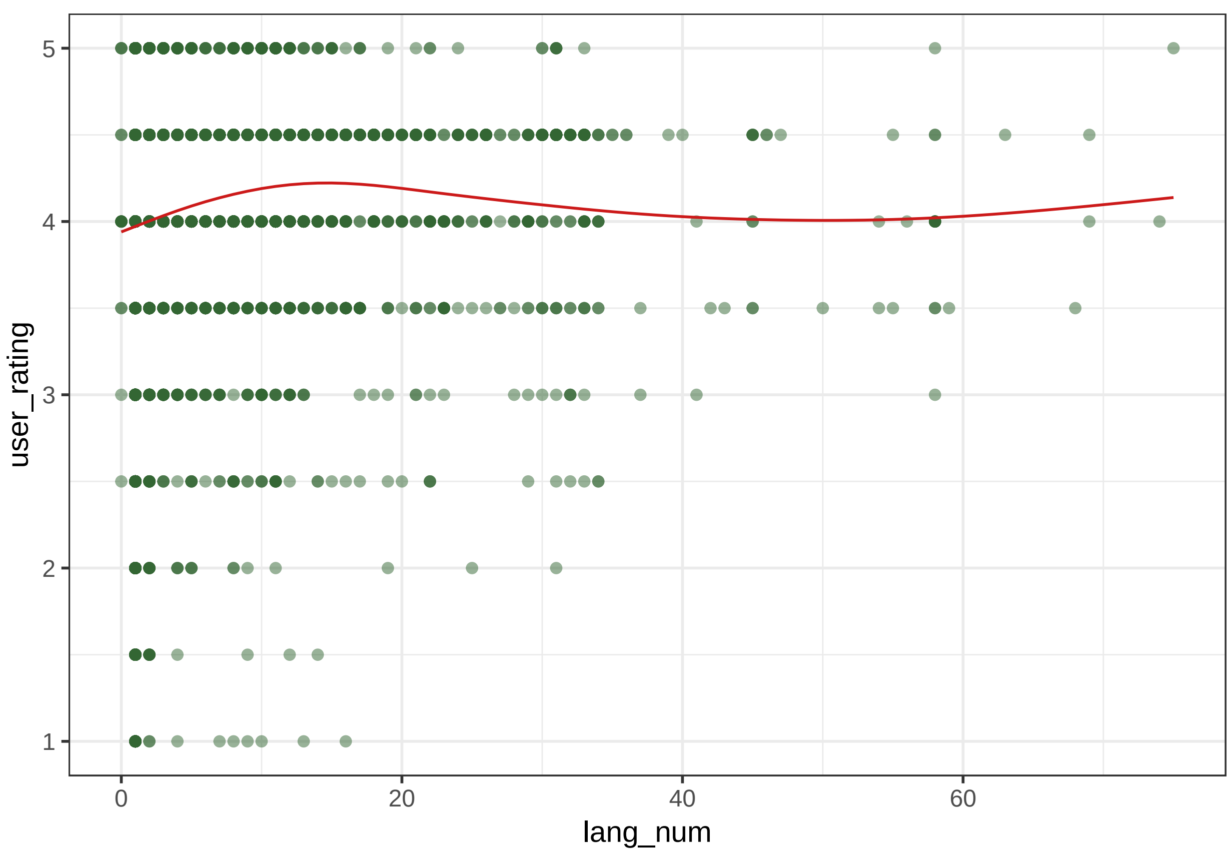
```
pred.ss <- predict(fit.ss, x = lang.grid) # specify x;
# but we did not calculate CI in this function
pred.ss.df <- data.frame(pred = pred.ss$y,
                         lang = lang.grid)
pred.ss.df
```

```
##        pred lang
## 1  3.939888    0
## 2  3.971486    1
```

```
## 3  4.002946    2
## 4  4.033415    3
## 5  4.062281    4
## 6  4.089302    5
## 7  4.114156    6
## 8  4.136734    7
## 9  4.157025    8
## 10 4.174931    9
## 11 4.190196   10
## 12 4.202575   11
## 13 4.211989   12
## 14 4.218362   13
## 15 4.221709   14
## 16 4.222136   15
## 17 4.219856   16
## 18 4.215214   17
## 19 4.208566   18
## 20 4.200341   19
## 21 4.191000   20
## 22 4.181011   21
## 23 4.170764   22
## 24 4.160606   23
## 25 4.150708   24
## 26 4.141051   25
## 27 4.131609   26
## 28 4.122356   27
## 29 4.113323   28
## 30 4.104544   29
## 31 4.095985   30
## 32 4.087553   31
## 33 4.079261   32
## 34 4.071252   33
## 35 4.063619   34
## 36 4.056443   35
## 37 4.049733   36
## 38 4.043514   37
## 39 4.037801   38
## 40 4.032598   39
## 41 4.027909   40
## 42 4.023734   41
## 43 4.020087   42
## 44 4.016956   43
## 45 4.014299   44
## 46 4.012058   45
## 47 4.010178   46
## 48 4.008635   47
## 49 4.007457   48
## 50 4.006673   49
## 51 4.006309   50
## 52 4.006392   51
## 53 4.006948   52
## 54 4.008004   53
## 55 4.009586   54
## 56 4.011711   55
```

```
## 57 4.014369   56
## 58 4.017553   57
## 59 4.021259   58
## 60 4.025476   59
## 61 4.030188   60
## 62 4.035357   61
## 63 4.040945   62
## 64 4.046912   63
## 65 4.053221   64
## 66 4.059849   65
## 67 4.066776   66
## 68 4.073981   67
## 69 4.081445   68
## 70 4.089143   69
## 71 4.097037   70
## 72 4.105093   71
## 73 4.113281   72
## 74 4.121567   73
## 75 4.129920   74
## 76 4.138309   75
```

```r
ggplot(data = train_data, aes(x = lang_num, y = user_rating)) +
     geom_point(color = rgb(.2, .4, .2, .5))+
geom_line(aes(x = lang, y = pred), data = pred.ss.df,
          color = rgb(.8, .1, .1, 1)) + theme_bw()
```



fit smoothing spline to user_rating_ver

```r
fit.ss <- smooth.spline(train_data$user_rating_ver, # predictor (univariate)
                        train_data$user_rating)  # response
fit.ss$df
```

```
## [1] 8.548232
```

```r
fit.ss$lambda # 0.05
```

```
## [1] 0.004566497
```

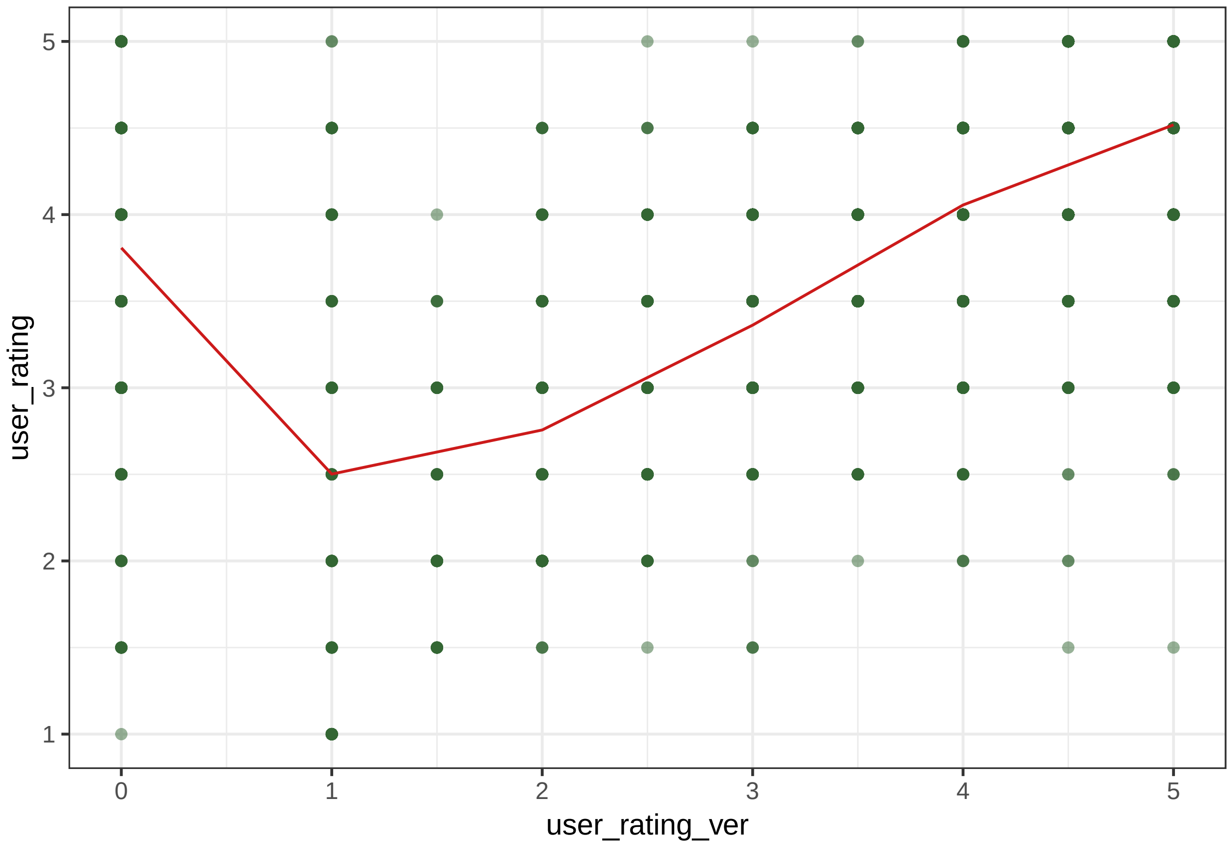
```r
# look at the range
ratelims <- range(train_data$user_rating_ver)

# create a sequence of observations
rate.grid <- seq(from = ratelims[1],to = ratelims[2])
rate.grid
```

```
## [1] 0 1 2 3 4 5
```

```r
pred.ss <- predict(fit.ss, x = rate.grid) # specify x;
# but we did not calculate CI in this function
pred.ss.df <- data.frame(pred = pred.ss$y,
                         rate = rate.grid)
pred.ss.df
```

```
##       pred rate
## 1 3.807699    0
## 2 2.501594    1
## 3 2.756369    2
## 4 3.361577    3
## 5 4.055344    4
## 6 4.518176    5
```

```r
ggplot(data = train_data, aes(x = user_rating_ver, y = user_rating)) +
     geom_point(color = rgb(.2, .4, .2, .5))+
geom_line(aes(x = rate, y = pred), data = pred.ss.df,
          color = rgb(.8, .1, .1, 1)) + theme_bw()
```



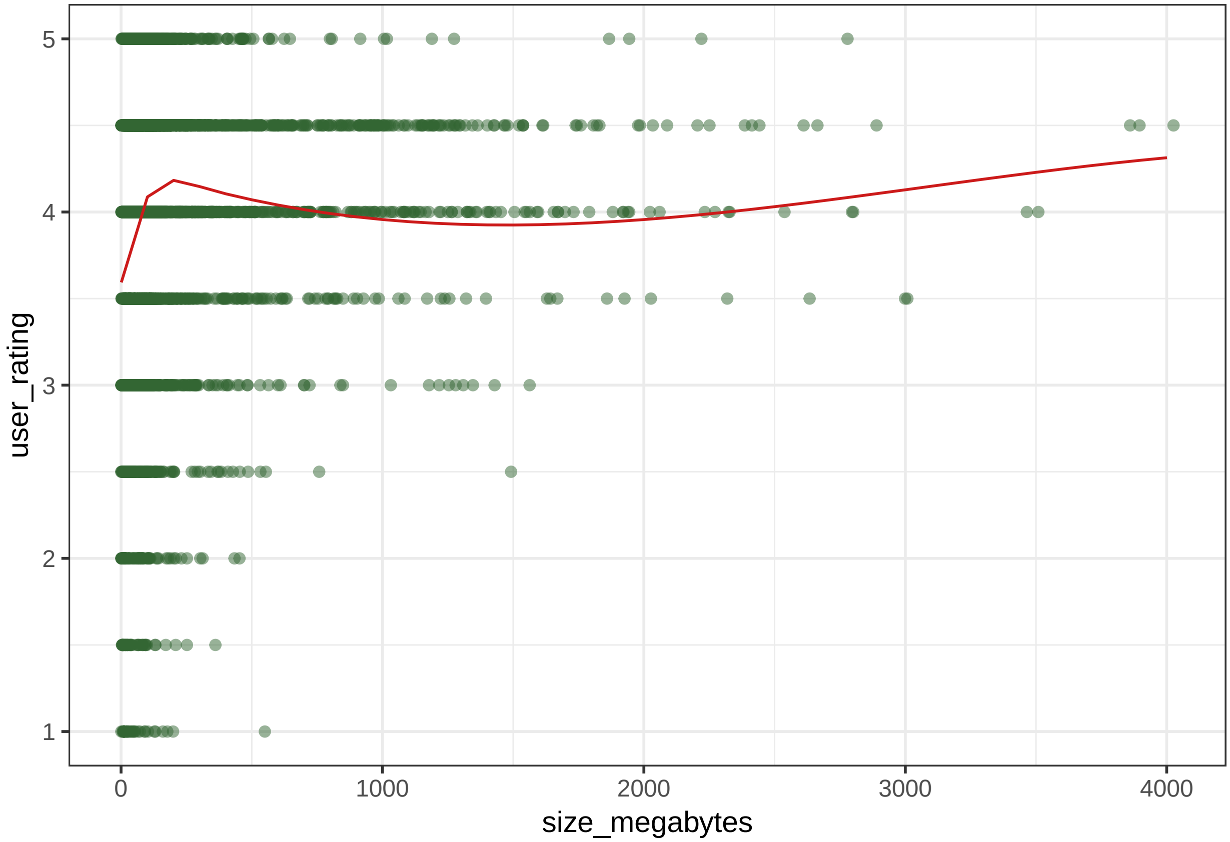
local regression

```
fit.loess <- loess(user_rating ~ size_megabytes, data = train_data)
summary(fit.loess)
```

```
## Call:
## loess(formula = user_rating ~ size_megabytes, data = train_data)
##
## Number of Observations: 4702
## Equivalent Number of Parameters: 5.67
## Residual Standard Error: 0.7161
## Trace of smoother matrix: 6.21  (exact)
##
## Control settings:
##   span     :  0.75
##   degree   :  2
##   family   :  gaussian
##   surface  :  interpolate      cell = 0.2
##   normalize:  TRUE
##  parametric:  FALSE
## drop.square:  FALSE
```

```
pred.loess <- predict(fit.loess, newdata = data.frame(size_megabytes = size.grid))

pred.loess.df <- data.frame(pred = pred.loess,
                            size = size.grid)
```

```r
p + geom_line(aes(x = size, y = pred), data = pred.loess.df,
              color = rgb(.8, .1, .1, 1)) + theme_bw()
```



GAM

mgcv package

```r
# Start with linear model; do not assume nonlinear trait
gam.m1 <- gam(user_rating ~ size_megabytes + price +
                user_rating_ver + cont_rating + prime_genre + sup_devices_num +
                ipad_sc_urls_num + lang_num , data = train_data)
summary(gam.m1)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## user_rating ~ size_megabytes + price + user_rating_ver + cont_rating +
##     prime_genre + sup_devices_num + ipad_sc_urls_num + lang_num
##
## Parametric coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          2.963e+00  9.958e-02  29.759  < 2e-16 ***
## size_megabytes      -5.369e-05  2.998e-05  -1.791  0.07338 .
## price                4.426e-03  1.845e-03   2.399  0.01650 *
## user_rating_ver      2.367e-01  6.893e-03  34.335  < 2e-16 ***
```

```
## cont_rating17+    3.643e-02  4.157e-02   0.876  0.38081
## cont_rating4+    -5.827e-02  2.610e-02  -2.232  0.02564 *
## cont_rating9+    -7.236e-03  3.407e-02  -0.212  0.83183
## prime_genre       1.352e-01  2.217e-02   6.099 1.16e-09 ***
## sup_devices_num   1.577e-03  2.455e-03   0.643  0.52056
## ipad_sc_urls_num  2.144e-02  5.310e-03   4.037 5.50e-05 ***
## lang_num          4.043e-03  1.115e-03   3.627  0.00029 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## R-sq.(adj) =  0.255   Deviance explained = 25.7%
## GCV = 0.3956  Scale est. = 0.39468   n = 4702
```

```
# add one non-linear component to size bytes
gam.m2 <- gam(user_rating ~ s(size_megabytes) + price +
                user_rating_ver + cont_rating + prime_genre + sup_devices_num +
                ipad_sc_urls_num + lang_num, data = train_data)
summary(gam.m2)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## user_rating ~ s(size_megabytes) + price + user_rating_ver + cont_rating +
##     prime_genre + sup_devices_num + ipad_sc_urls_num + lang_num
##
## Parametric coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       2.978159   0.098589  30.208  < 2e-16 ***
## price             0.005050   0.001846   2.736  0.00625 **
## user_rating_ver   0.236489   0.006869  34.430  < 2e-16 ***
## cont_rating17+    0.027024   0.041502   0.651  0.51499
## cont_rating4+    -0.057664   0.026054  -2.213  0.02693 *
## cont_rating9+    -0.006744   0.034191  -0.197  0.84364
## prime_genre       0.117905   0.022467   5.248 1.61e-07 ***
## sup_devices_num   0.001682   0.002449   0.687  0.49211
## ipad_sc_urls_num  0.017396   0.005380   3.234  0.00123 **
## lang_num          0.003257   0.001120   2.907  0.00366 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                     edf Ref.df     F  p-value
## s(size_megabytes) 7.708  8.544 4.853 2.94e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.261   Deviance explained = 26.3%
## GCV = 0.39312  Scale est. = 0.39164   n = 4702
```

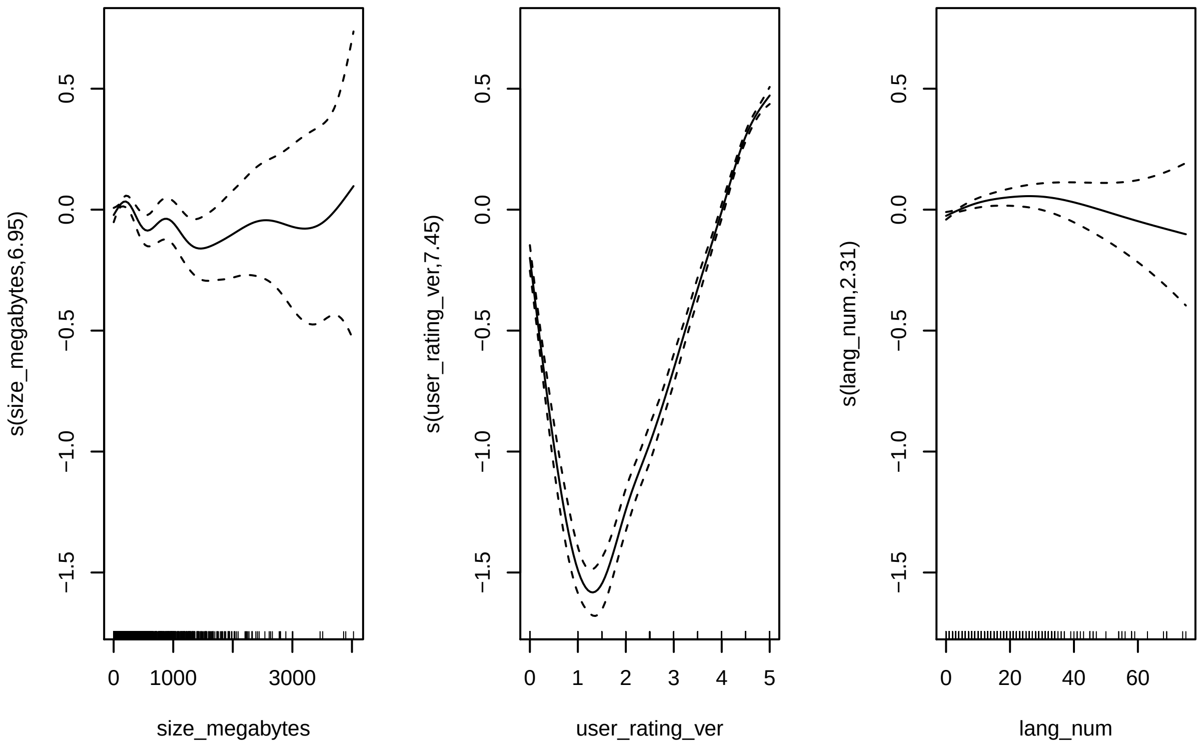
```
#  add one non-linear component to lang_num
gam.m3 <- gam(user_rating ~ s(size_megabytes) + price +
                user_rating_ver + cont_rating + prime_genre + sup_devices_num +
```

```r
                 ipad_sc_urls_num + s(lang_num), data = train_data)

# add one non-linear component to user rating of current version
gam.m4 <- gam(user_rating ~ s(size_megabytes) + price +
                 s(user_rating_ver) + cont_rating + prime_genre + sup_devices_num +
                 ipad_sc_urls_num + s(lang_num), data = train_data)

anova(gam.m1, gam.m2, gam.m3, gam.m4, test = "F")
```

```
## Analysis of Deviance Table
##
## Model 1: user_rating ~ size_megabytes + price + user_rating_ver + cont_rating +
##     prime_genre + sup_devices_num + ipad_sc_urls_num + lang_num
## Model 2: user_rating ~ s(size_megabytes) + price + user_rating_ver + cont_rating +
##     prime_genre + sup_devices_num + ipad_sc_urls_num + lang_num
## Model 3: user_rating ~ s(size_megabytes) + price + user_rating_ver + cont_rating +
##     prime_genre + sup_devices_num + ipad_sc_urls_num + s(lang_num)
## Model 4: user_rating ~ s(size_megabytes) + price + s(user_rating_ver) +
##     cont_rating + prime_genre + sup_devices_num + ipad_sc_urls_num +
##     s(lang_num)
##   Resid. Df Resid. Dev     Df Deviance        F    Pr(>F)
## 1    4691.0     1851.4
## 2    4683.5     1834.5 7.5443    16.88   8.2687 1.128e-10 ***
## 3    4679.3     1825.1 4.1217     9.44   8.4601 6.081e-07 ***
## 4    4674.9     1265.9 4.4536   559.17 463.8916 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

plot smoothing component

```r
par(mfrow = c(1,3))
plot(gam.m4)
```



use caret package to do gam

```
set.seed(1234)
gam.fit <- train(x_train, y_train,
                 method = "gam", # use gam
                 tuneGrid = data.frame(method = "GCV.Cp", select = c(TRUE,FALSE)),
                 # two tuning parameters (but not real tuning paramteres from what we taught in class)
                 # select = c(to do feature selection or not(set coeffcients to be zero or not), method
                 trControl = ctrl1)

gam.fit$bestTune # by doing feature selection, get a better error in terms of MSE

gam.fit$finalModel
```

MARS

```
library(pdp)
```

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

```
library(earth)
```

```
## Warning: package 'earth' was built under R version 3.5.2
## Loading required package: Formula
## Loading required package: plotmo
```

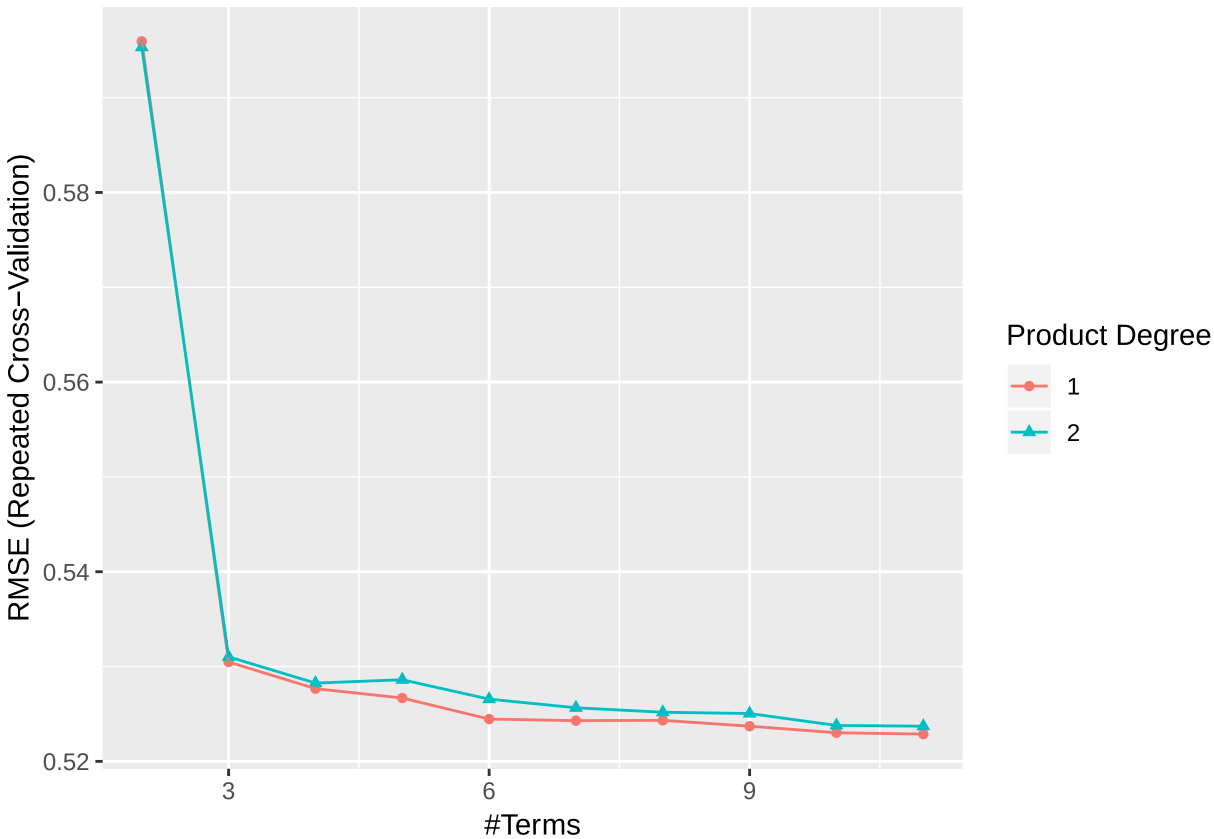
```
## Loading required package: plotrix
```

```
## Loading required package: TeachingDemos
```

```r
mars_grid <- expand.grid(degree = 1:2,  # to include interaction or not
                         nprune = 2:11) # how many variables you want to include
mars_grid
```

```
##    degree nprune
## 1       1      2
## 2       2      2
## 3       1      3
## 4       2      3
## 5       1      4
## 6       2      4
## 7       1      5
## 8       2      5
## 9       1      6
## 10      2      6
## 11      1      7
## 12      2      7
## 13      1      8
## 14      2      8
## 15      1      9
## 16      2      9
## 17      1     10
## 18      2     10
## 19      1     11
## 20      2     11
```

```r
set.seed(1234)
mars.fit <- train(x_train, y_train,
                  method = "earth",
                  tuneGrid = mars_grid,
                  trControl = ctrl1)

ggplot(mars.fit) # each line is for one degree;
```




```r
mars.fit$bestTune # model contains 3 variables with interaction terms
```

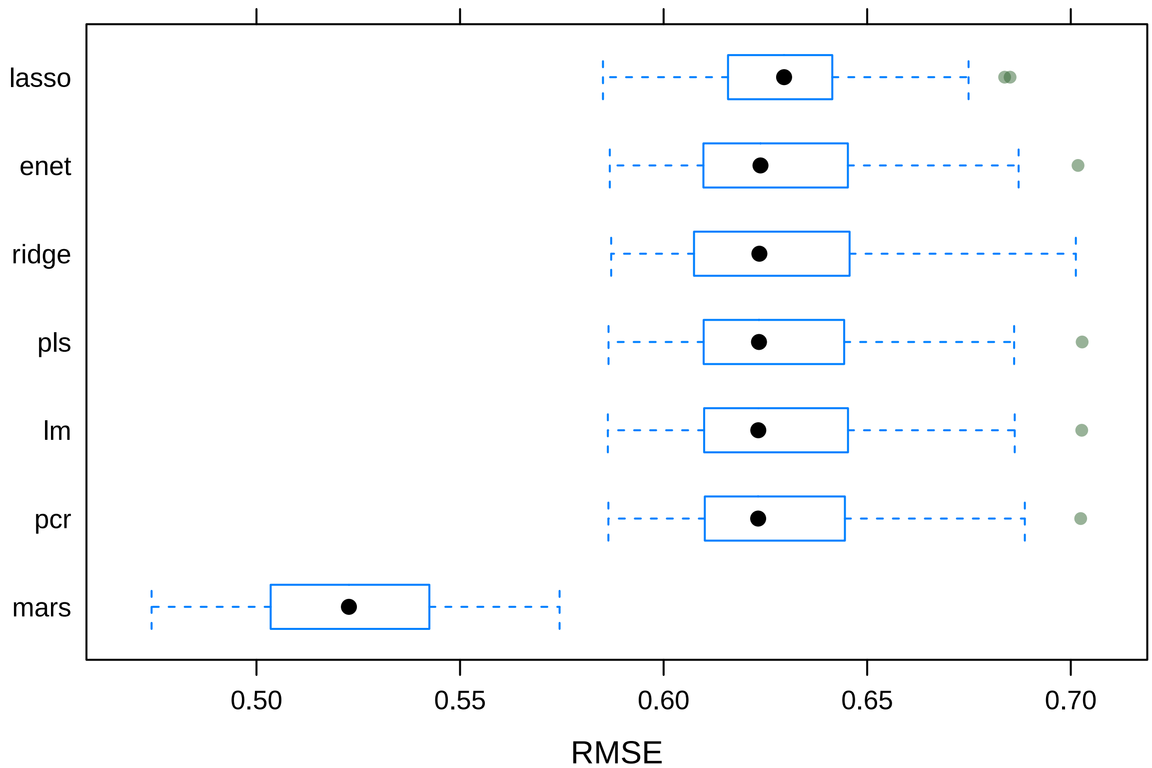
```
##    nprune degree
## 10     11      1
```

```r
coef(mars.fit$finalModel)
```

```
##            (Intercept)   h(user_rating_ver-1)   h(1-user_rating_ver)
##            2.993630649            0.280965560            1.354065765
##            prime_genre h(user_rating_ver-4.5)   h(user_rating_ver-2)
##            0.118120452           -0.303267163            0.348398190
##          h(30-lang_num)   h(size_megabytes-118)  h(118-size_megabytes)
##           -0.003328511            0.005062028           -0.004994209
##   h(size_megabytes-44)          h(5.99-price)
##           -0.005147150           -0.015408822
```

Compare models

```r
#jpeg("rmse.jpg")
bwplot(resamples(list(#gam = gam.fit,
                      lm = lm.fit,
                      mars = mars.fit,
                      ridge = ridge.fit,
                      lasso = lasso.fit,
                      enet = enet.fit,
                      pcr = pcr.fit,
                      pls = pls.fit)), metric = "RMSE")
```




```r
#dev.off()
```

```r
summary(resamples(list(#gam = gam.fit,
                       lm = lm.fit,
                       mars = mars.fit,
                       ridge = ridge.fit,
                       lasso = lasso.fit,
                       enet = enet.fit,
                       pcr = pcr.fit,
                       pls = pls.fit)), metric = "RMSE")
```

```
##
## Call:
## summary.resamples(object = resamples(list(lm = lm.fit, mars =
##  mars.fit, ridge = ridge.fit, lasso = lasso.fit, enet = enet.fit, pcr
##  = pcr.fit, pls = pls.fit)), metric = "RMSE")
##
## Models: lm, mars, ridge, lasso, enet, pcr, pls
## Number of resamples: 50
##
## RMSE
##            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## lm    0.5863014 0.6117991 0.6232496 0.6289451 0.6451086 0.7026959    0
## mars  0.4742350 0.5048572 0.5227044 0.5228728 0.5415354 0.5744478    0
## ridge 0.5871200 0.6096714 0.6235217 0.6289694 0.6453525 0.7012491    0
## lasso 0.5851028 0.6158286 0.6295917 0.6290490 0.6413175 0.6850979    0
## enet  0.5867800 0.6113575 0.6237916 0.6287702 0.6451504 0.7017853    0
## pcr   0.5864340 0.6108432 0.6232179 0.6283084 0.6441235 0.7024380    0
## pls   0.5864686 0.6117803 0.6234221 0.6288189 0.6442827 0.7027942    0
```

Test MSE

```
pred.elast <- predict(enet.fit$finalModel, newx = x_test, s = enet.fit$bestTune$lambda, type = "respons
mean((pred.elast - y_test)^2) # 0.407
```

```
## [1] 0.4069269
```

```
pred.mars <- predict(mars.fit$finalModel, newdata = x_test, type = "response")
mean((pred.mars - y_test)^2) # 0.283
```

```
## [1] 0.2830337
```

```
pred.lasso <- predict(lasso.fit$finalModel, newx = x_test, s = lasso.fit$bestTune$lambda, type = "respo
mean((pred.lasso - y_test)^2) # 0.407
```

```
## [1] 0.407003
```

```
pred.pcr <- predict(pcr.fit$finalModel, newdata = x_test, ncomp = pcr.fit$bestTune[[1]], type = "respon
mean((pred.pcr - y_test)^2) # 0.407
```

```
## [1] 0.4068258
```

```
pred.pls <- predict(pls.fit$finalModel, newdata = x_test, ncomp = pls.fit$bestTune$ncomp, type = "respo
mean((pred.pls - y_test)^2) # 0.406
```

```
## [1] 0.4065176
```

```
#pred.lm <- predict(lm.fit$finalModel, newdata = x_test)
#mean((pred.lm - y_test)^2) # 0.41??
pred.ridge <- predict(ridge.fit$finalModel, newx = x_test, s = ridge.fit$bestTune$lambda, type = "respo
mean((pred.ridge - y_test)^2) # 0.406
```

```
## [1] 0.4061881
```