Homework 4: Moving Beyond Linearity

Wanitchaya Poonpatanapricha

Non-linear regression

```
set.seed(232) # set seed

# load data
gss_train <- read.csv("data/gss_train.csv")
gss_test <- read.csv("data/gss_test.csv")

k <- 10 # 10 fold cv
fold <- sample(k, nrow(gss_train), replace = TRUE) # save folds
```

1) Polynomial Regression

```
# initialize data structure to hold CV MSE
mse <- numeric(k)
span <- seq(1, 10, by = 1)
cv <- numeric(length(span))
ame <- numeric(k)
ame_cv <- numeric(length(span))

for (j in seq_along(span)){
  for (i in seq_len(k)){
    take <- fold == i
    foldi <- gss_train[take, ]
    foldOther <- gss_train[!take, ]
    # fit polynomial of j degree
    f <- lm(egalit_scale ~ poly(income06, span[j]), data=foldOther)
    pred <- predict(f, foldi)
    # mse for one fold
    mse[i] <- mean((pred - foldi$egalit_scale)^2, na.rm = TRUE)}
  # mse across 10 folds
  cv[j]<- mean(mse)}

# choose best degree
df_poly <- cbind(as.data.frame(span), as.data.frame(cv))
names(df_poly) <- c("Degree", "CV MSE")
df_poly %>%
  arrange(cv) %>%
  head() %>%
  kable(caption = "Top Polynomial Degree by CV MSE")
```

Table 1: Top Polynomial Degree by CV MSE

Degree	CV MSE
2	87.12
9	87.22
10	87.24
3	87.28
8	87.41
1	87.41

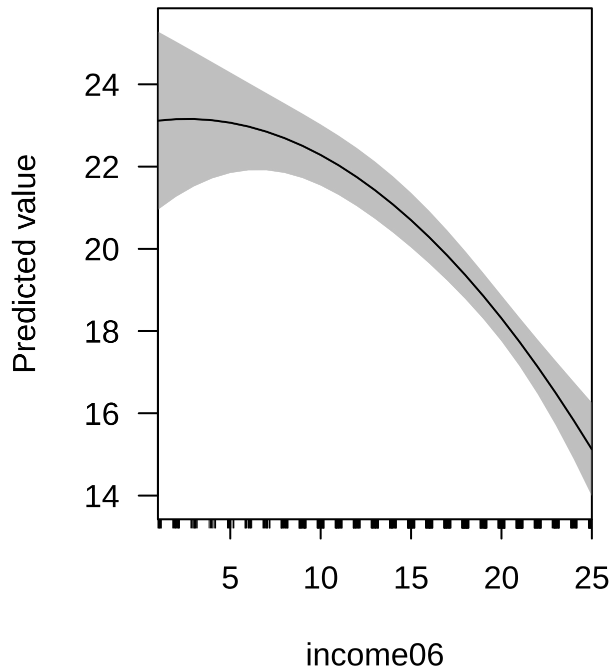
From the table, the optimal degree for the polynomial regression is 2.

```r
# fir 2nd degree polynomial model
best_poly <- lm(egalit_scale ~ income06 + I(income06^2), data = gss_train)
# AME
margins(best_poly)
```

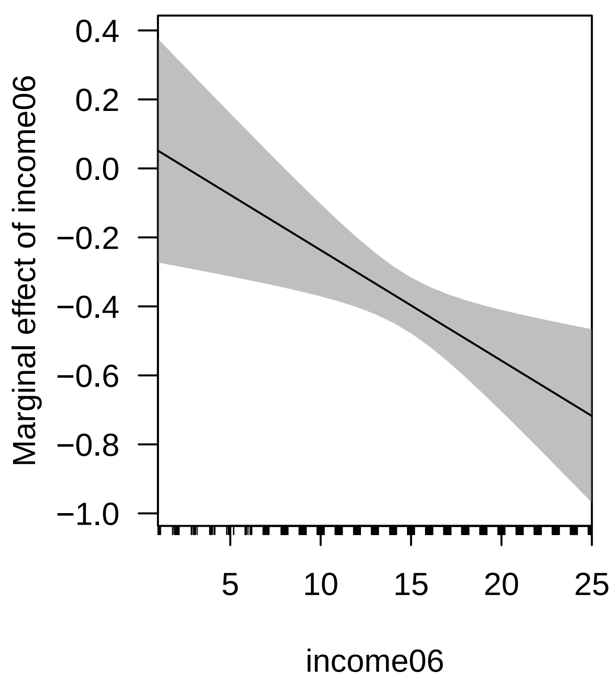
```
##  income06
##   -0.4507
```

```r
# predicted value and marginal effect
cplot(best_poly, "income06")
```

```
##    xvals yvals upper lower
## 1      1 23.12 25.28 20.95
## 2      2 23.15 25.04 21.26
## 3      3 23.16 24.79 21.52
## 4      4 23.13 24.54 21.71
## 5      5 23.07 24.29 21.84
## 6      6 22.97 24.04 21.91
## 7      7 22.85 23.79 21.91
## 8      8 22.69 23.54 21.84
## 9      9 22.50 23.29 21.72
## 10    10 22.28 23.03 21.54
## 11    11 22.03 22.75 21.31
## 12    12 21.74 22.45 21.04
## 13    13 21.43 22.13 20.73
## 14    14 21.08 21.76 20.39
## 15    15 20.70 21.36 20.03
## 16    16 20.28 20.93 19.64
## 17    17 19.84 20.45 19.23
## 18    18 19.36 19.94 18.78
## 19    19 18.85 19.41 18.29
## 20    20 18.31 18.87 17.75
```




```r
cplot(best_poly, "income06", what = "effect")
```



Since the model is 2nd degree polynomial, we can see a curvilinear prediction line with one local extrema (maxima in this case), and we can see a linear, negative slope of marginal effect. This means that as `income06` increases, the marginal effect of `income06` on the predicted value decreases from slightly positive marginal effect to strongly negative marginal effect eventually. The AME for `income06` is -0.4507.

2) Step Function

```r
# structure to store CV MSE
cv.error <- rep (0,10)
# choosing best steps
for (i in 1:10){
  labs <- levels(cut(gss_train$income06, i+1))
  breaks <- unique(c(as.numeric(sub("\\((.+),.*", "\\1", labs)),
                    as.numeric(sub("[^,]*,([^]]*)\\]", "\\1", labs))))
  # step function with i+1 steps
  step.fit <- glm(egalit_scale~cut(income06,unique(breaks)), data = gss_train)
  # 10 CV
  cv.error[i] <- cv.glm(gss_train ,step.fit, K=10)$delta[1]
}
cv.error <- as.data.frame(cv.error)
names(cv.error) <- c("mse")
cv.error$n_step <- c(2:11)
cv.error %>%
  dplyr::select(n_step, mse) %>%
  arrange(mse) %>%
  head() %>%
  kable(caption = "Top number of steps by CV MSE")
```

Table 2: Top number of steps by CV MSE

n_step	mse
4	87.56
9	87.64
8	87.82
7	87.84
11	88.04
5	88.07

From the table, the optimal number of steps/bins is 4.

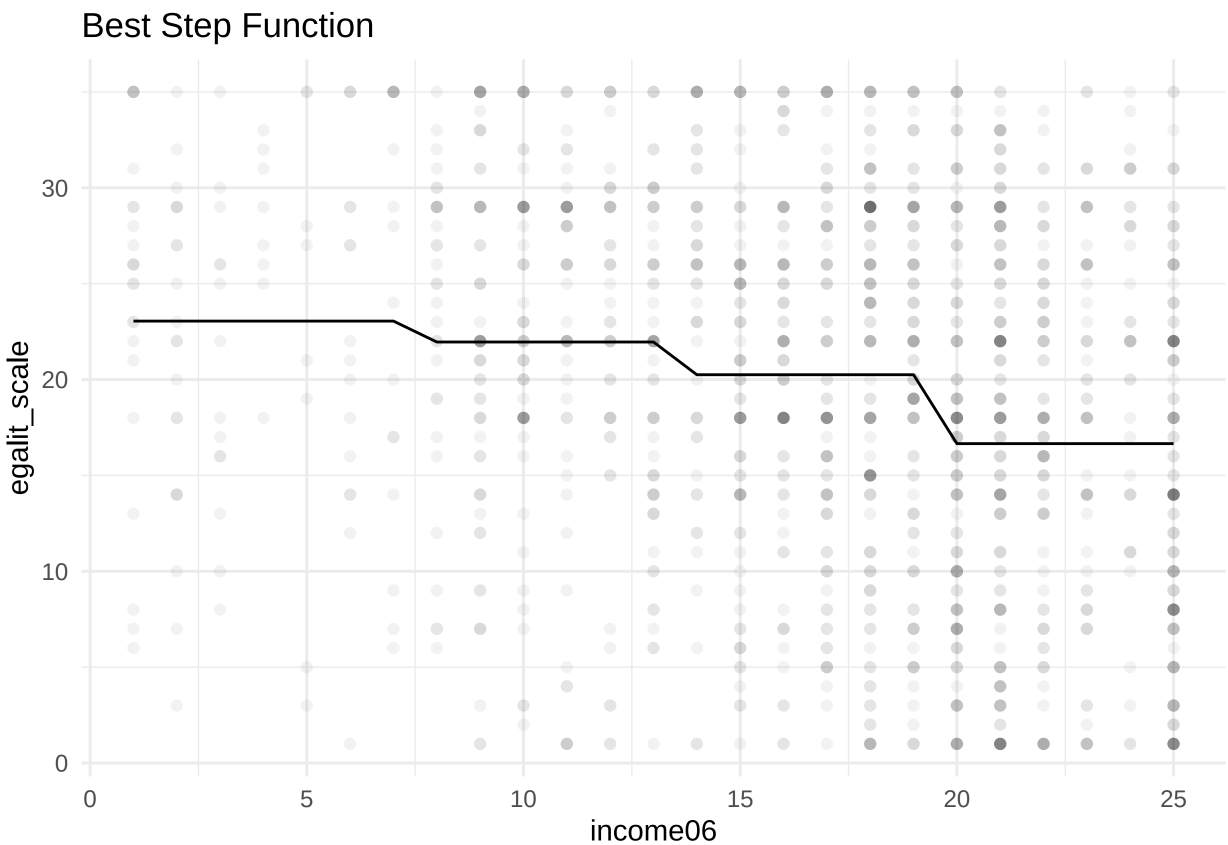
```r
# choose the best number of steps
best_step <- cv.error %>% arrange(mse)
best_step_n <- best_step[1,]$n_step

# fit best step function
labs <- levels(cut(gss_train$income06, best_step_n))
breaks <- unique(c(as.numeric(sub("\\((.+),.*", "\\1", labs)),
                  as.numeric(sub("[^,]*,([^]]*)\\]", "\\1", labs))))
step.fit <- glm(egalit_scale~cut(income06,unique(breaks)), data = gss_train)

# predict the values
pred_step <- predict(step.fit, gss_train)
df_step <- cbind(gss_train$income06, pred_step) %>%
  as.data.frame()

# plot best step function against actual values
ggplot() +
  geom_point(data = gss_train, aes(income06, egalit_scale), alpha = 0.05) +
```

```r
  geom_line(data = df_step, aes(V1, pred_step)) +
  labs(title = "Best Step Function")
```



We can clearly see the 4 bins of `income06`. In general, the predicted value decreases as `income06` decreases.

3) Natural Cubic Spline

```r
# function to fit one natural spline
vote_spline <- function(splits, df = NULL){
  model <- glm(egalit_scale ~ ns(income06, df = df),
               data = analysis(splits))
  model_mse <- augment(model, newdata = assessment(splits)) %>%
    mse(truth = egalit_scale, estimate = .fitted)
  mean(model_mse$.estimate)
}

# specify natural cubic spline
tune_over_knots <- function(splits, knots){
  vote_spline(splits, df = knots + 3)
}

# cv splits
results <- vfold_cv(gss_train, v = 10)

# estimate 1 to 8 knots
best_knot <- tidyr::expand(results, id, knots = 1:8) %>%
```

```
  left_join(results) %>%
  mutate(mse = map2_dbl(splits, knots, tune_over_knots)) %>%
  group_by(knots) %>%
  summarize(mse = mean(mse)) %>%
  arrange(mse)

best_knot %>%
  head() %>%
  kable(caption = "Top number of knots by CV MSE")
```

Table 3: Top number of knots by CV MSE

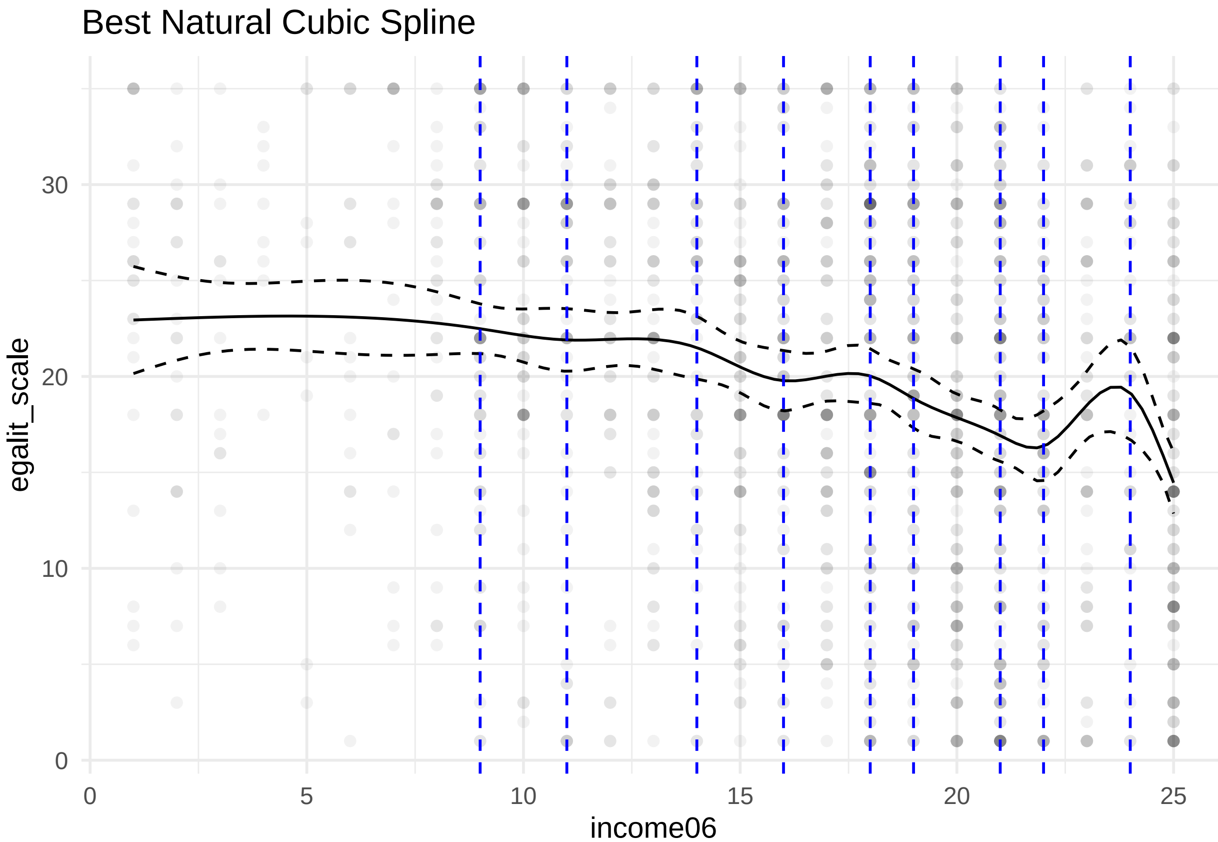
knots	mse
7	87.53
4	87.57
1	87.62
8	87.67
3	87.69
5	87.70

From the table, the optimal number of knots is 7 knots.

```
# df = 7 + 3 = 10
glm(egalit_scale ~ ns(income06, df = 10), data = gss_train) %>%
  cplot("income06", what = "prediction", n = 100, draw = FALSE) %>%
  ggplot(aes(x = xvals)) +
  geom_line(aes(y = yvals)) + # predicted value
  geom_line(aes(y = upper), linetype = 2) + # upper SE
  geom_line(aes(y = lower), linetype = 2) + # lower SE
  geom_point(data = gss_train, aes(income06, egalit_scale), alpha = 0.05) +
  geom_vline(xintercept = attr(ns(gss_train$income06, df = 10), "knots"),
             linetype = 2, color = "blue") +
  labs(x = "income06",
       y = "egalit_scale",
       title = "Best Natural Cubic Spline")
```

```
##    xvals yvals upper lower
## 1  1.000 22.94 25.73 20.15
## 2  1.242 22.96 25.59 20.34
## 3  1.485 22.98 25.45 20.51
## 4  1.727 23.00 25.33 20.68
## 5  1.970 23.02 25.22 20.83
## 6  2.212 23.04 25.12 20.97
## 7  2.455 23.06 25.03 21.09
## 8  2.697 23.08 24.96 21.19
## 9  2.939 23.09 24.91 21.28
## 10 3.182 23.11 24.87 21.34
## 11 3.424 23.12 24.85 21.39
## 12 3.667 23.13 24.85 21.41
## 13 3.909 23.14 24.85 21.42
## 14 4.152 23.14 24.87 21.42
## 15 4.394 23.15 24.90 21.40
## 16 4.636 23.15 24.92 21.37
```

```
## 17 4.879 23.15 24.95 21.34
## 18 5.121 23.14 24.98 21.30
## 19 5.364 23.13 25.00 21.26
## 20 5.606 23.12 25.01 21.23
```



From the plot, we can see the negative trend of `income06` on the predicted value as in polynomial regression and in step function. However, the model is a lot more flexible (and smoother than step function). The interval around the predicted value is consistent and small across `income06`. This is desirable.

4) Egalitarianism and everything (Models Fitting and Tuning)

```
# standardize numeric features (scale and center)
standard <- function(data){
  df <- data %>%
    mutate_if(is.numeric, scale) %>%
    mutate_if(is.numeric, c)
}
gss_train <- standard(gss_train)
gss_test <- standard(gss_test)

# create trainControl for 10-fold cv
cv_10 = trainControl(method = "cv", number = 10)
```

a) Linear regression

NOTE: There aren't anything to tune for linear regression. However, I fit both train and test data to match the other 3 models where I tune the models using 10-folds cv on train data and fit the model with the best hyperparameters on test data (according to piazza post @71). I also use the scaled data (even though it is not necessary for linear regression) so that I can compare the (R)MSE across models.

```
# train model
lm_caret <- train(
  egalit_scale ~ ., gss_train,
  method = "lm",
  trControl = cv_10
  )
lm_caret
```

```
## Linear Regression
##
## 1481 samples
##   44 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1334, 1333, 1332, 1332, 1333, 1334, ...
## Resampling results:
##
##   RMSE    Rsquared  MAE
##   0.8274  0.3232    0.6543
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
# test model
lm_caret <- train(
  egalit_scale ~ ., gss_test,
  method = "lm"
  )
lm_caret
```

```
## Linear Regression
##
## 493 samples
##  44 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 493, 493, 493, 493, 493, 493, ...
## Resampling results:
##
##   RMSE   Rsquared  MAE
##   1.006  0.1764    0.7972
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

The train MSE is 0.6846 and the final MSE is 1.012.

b) Elastic net regression

```
# find best alpha and lambda
gss_elnet = train(
  egalit_scale ~ ., data = gss_train,
  method = "glmnet",
  trControl = cv_10,
  tuneLength = 10
)
gss_elnet
```

```
## glmnet
##
## 1481 samples
##   44 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1333, 1332, 1334, 1332, 1334, 1332, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda     RMSE    Rsquared  MAE
##   0.1    0.0002201  0.8230  0.3303    0.6491
##   0.1    0.0005086  0.8230  0.3303    0.6491
##   0.1    0.0011748  0.8226  0.3308    0.6490
##   0.1    0.0027140  0.8216  0.3320    0.6484
##   0.1    0.0062697  0.8203  0.3336    0.6474
##   0.1    0.0144837  0.8178  0.3364    0.6458
##   0.1    0.0334593  0.8139  0.3411    0.6441
##   0.1    0.0772954  0.8096  0.3471    0.6439
##   0.1    0.1785624  0.8082  0.3537    0.6481
##   0.1    0.4125024  0.8243  0.3453    0.6683
##   0.2    0.0002201  0.8231  0.3302    0.6492
##   0.2    0.0005086  0.8228  0.3306    0.6491
##   0.2    0.0011748  0.8221  0.3315    0.6487
##   0.2    0.0027140  0.8209  0.3329    0.6478
##   0.2    0.0062697  0.8189  0.3353    0.6463
##   0.2    0.0144837  0.8153  0.3396    0.6444
##   0.2    0.0334593  0.8108  0.3453    0.6428
##   0.2    0.0772954  0.8061  0.3540    0.6435
##   0.2    0.1785624  0.8150  0.3482    0.6569
##   0.2    0.4125024  0.8507  0.3152    0.6962
##   0.3    0.0002201  0.8230  0.3303    0.6491
##   0.3    0.0005086  0.8227  0.3308    0.6490
##   0.3    0.0011748  0.8216  0.3321    0.6483
##   0.3    0.0027140  0.8202  0.3338    0.6473
##   0.3    0.0062697  0.8175  0.3370    0.6455
##   0.3    0.0144837  0.8134  0.3421    0.6433
##   0.3    0.0334593  0.8087  0.3486    0.6427
##   0.3    0.0772954  0.8067  0.3549    0.6458
##   0.3    0.1785624  0.8248  0.3377    0.6680
##   0.3    0.4125024  0.8730  0.2891    0.7194
##   0.4    0.0002201  0.8230  0.3303    0.6491
##   0.4    0.0005086  0.8225  0.3311    0.6489
```

```
##   0.4    0.0011748  0.8213  0.3325    0.6481
##   0.4    0.0027140  0.8196  0.3345    0.6468
##   0.4    0.0062697  0.8164  0.3385    0.6448
##   0.4    0.0144837  0.8119  0.3441    0.6425
##   0.4    0.0334593  0.8066  0.3522    0.6423
##   0.4    0.0772954  0.8102  0.3509    0.6502
##   0.4    0.1785624  0.8378  0.3199    0.6818
##   0.4    0.4125024  0.8932  0.2616    0.7390
##   0.5    0.0002201  0.8229  0.3305    0.6491
##   0.5    0.0005086  0.8222  0.3314    0.6487
##   0.5    0.0011748  0.8210  0.3329    0.6479
##   0.5    0.0027140  0.8190  0.3353    0.6463
##   0.5    0.0062697  0.8154  0.3397    0.6442
##   0.5    0.0144837  0.8106  0.3457    0.6421
##   0.5    0.0334593  0.8054  0.3546    0.6422
##   0.5    0.0772954  0.8138  0.3469    0.6547
##   0.5    0.1785624  0.8493  0.3042    0.6943
##   0.5    0.4125024  0.9086  0.2419    0.7543
##   0.6    0.0002201  0.8229  0.3305    0.6491
##   0.6    0.0005086  0.8219  0.3318    0.6485
##   0.6    0.0011748  0.8207  0.3333    0.6476
##   0.6    0.0027140  0.8184  0.3361    0.6459
##   0.6    0.0062697  0.8145  0.3409    0.6436
##   0.6    0.0144837  0.8096  0.3471    0.6420
##   0.6    0.0334593  0.8053  0.3554    0.6429
##   0.6    0.0772954  0.8179  0.3422    0.6594
##   0.6    0.1785624  0.8586  0.2922    0.7043
##   0.6    0.4125024  0.9194  0.2381    0.7654
##   0.7    0.0002201  0.8228  0.3307    0.6491
##   0.7    0.0005086  0.8217  0.3320    0.6484
##   0.7    0.0011748  0.8204  0.3337    0.6474
##   0.7    0.0027140  0.8178  0.3368    0.6455
##   0.7    0.0062697  0.8136  0.3419    0.6432
##   0.7    0.0144837  0.8088  0.3483    0.6422
##   0.7    0.0334593  0.8059  0.3549    0.6442
##   0.7    0.0772954  0.8232  0.3353    0.6654
##   0.7    0.1785624  0.8673  0.2807    0.7131
##   0.7    0.4125024  0.9315  0.2314    0.7769
##   0.8    0.0002201  0.8227  0.3308    0.6490
##   0.8    0.0005086  0.8216  0.3322    0.6483
##   0.8    0.0011748  0.8201  0.3340    0.6472
##   0.8    0.0027140  0.8172  0.3375    0.6452
##   0.8    0.0062697  0.8129  0.3429    0.6427
##   0.8    0.0144837  0.8079  0.3498    0.6421
##   0.8    0.0334593  0.8072  0.3534    0.6460
##   0.8    0.0772954  0.8290  0.3269    0.6717
##   0.8    0.1785624  0.8760  0.2682    0.7213
##   0.8    0.4125024  0.9429  0.2303    0.7871
##   0.9    0.0002201  0.8226  0.3309    0.6489
##   0.9    0.0005086  0.8214  0.3324    0.6482
##   0.9    0.0011748  0.8198  0.3343    0.6469
##   0.9    0.0027140  0.8168  0.3381    0.6449
##   0.9    0.0062697  0.8122  0.3437    0.6424
##   0.9    0.0144837  0.8070  0.3514    0.6419
```

```
##   0.9    0.0334593  0.8088  0.3515    0.6479
##   0.9    0.0772954  0.8351  0.3178    0.6781
##   0.9    0.1785624  0.8843  0.2555    0.7292
##   0.9    0.4125024  0.9558  0.2303    0.7977
##   1.0    0.0002201  0.8225  0.3311    0.6489
##   1.0    0.0005086  0.8213  0.3326    0.6481
##   1.0    0.0011748  0.8196  0.3346    0.6467
##   1.0    0.0027140  0.8163  0.3387    0.6446
##   1.0    0.0062697  0.8116  0.3445    0.6422
##   1.0    0.0144837  0.8061  0.3528    0.6417
##   1.0    0.0334593  0.8105  0.3495    0.6500
##   1.0    0.0772954  0.8404  0.3101    0.6838
##   1.0    0.1785624  0.8919  0.2433    0.7370
##   1.0    0.4125024  0.9710  0.2303    0.8119
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0.6 and lambda = 0.03346.
```

```r
# extract best alpha and lambda
myGrid <- expand.grid(alpha = gss_elnet$bestTune$alpha,
                      lambda = gss_elnet$bestTune$lambda)

# fit model with best alpha and lambda
gss_elnet_best = train(
  egalit_scale ~ ., data = gss_test,
  method = "glmnet",
  tuneGrid = myGrid
)
gss_elnet_best
```

```
## glmnet
##
## 493 samples
##  44 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 493, 493, 493, 493, 493, 493, ...
## Resampling results:
##
##   RMSE    Rsquared  MAE
##   0.8703  0.255     0.6929
##
## Tuning parameter 'alpha' was held constant at a value of 0.6
## Tuning
##  parameter 'lambda' was held constant at a value of 0.03346
```

After the tuning, the best alpha is 0.6 and the best lambda is 0.03346. The train CV MSE is 0.6485 and the final MSE is 0.7574. The final MSE is better that that of linear regression. This is expected because the regularization help lower the variance of the model.

c) PCR

```r
# find best ncomp
gss_pcr_caret = train(
  egalit_scale ~ ., data = gss_train,
  method = "pcr",
  trControl = cv_10,
  tuneLength = 100
)
gss_pcr_caret
```

```
## Principal Component Analysis
##
## 1481 samples
##   44 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1334, 1333, 1332, 1333, 1332, 1334, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE    Rsquared  MAE
##     1    0.9999  0.004262  0.8380
##     2    0.9387  0.122609  0.7759
##     3    0.9218  0.154569  0.7617
##     4    0.9177  0.162224  0.7591
##     5    0.9157  0.166403  0.7566
##     6    0.9158  0.166381  0.7563
##     7    0.9119  0.172625  0.7521
##     8    0.9119  0.172370  0.7504
##     9    0.9122  0.171796  0.7508
##    10    0.9120  0.172254  0.7495
##    11    0.8981  0.197811  0.7376
##    12    0.9001  0.194011  0.7387
##    13    0.8735  0.242062  0.7057
##    14    0.8741  0.241411  0.7054
##    15    0.8666  0.254762  0.6973
##    16    0.8655  0.256371  0.6964
##    17    0.8667  0.254618  0.6968
##    18    0.8647  0.258542  0.6956
##    19    0.8628  0.261564  0.6947
##    20    0.8619  0.262234  0.6937
##    21    0.8494  0.282818  0.6803
##    22    0.8480  0.285242  0.6813
##    23    0.8441  0.292420  0.6774
##    24    0.8447  0.291888  0.6771
##    25    0.8449  0.291366  0.6774
##    26    0.8446  0.291761  0.6748
##    27    0.8408  0.297040  0.6709
##    28    0.8383  0.300250  0.6689
##    29    0.8377  0.301672  0.6679
##    30    0.8380  0.301178  0.6676
##    31    0.8288  0.316191  0.6594
##    32    0.8264  0.320217  0.6581
```

```
##     33     0.8248  0.322620  0.6563
##     34     0.8275  0.318481  0.6585
##     35     0.8257  0.321141  0.6574
##     36     0.8240  0.323306  0.6557
##     37     0.8227  0.325203  0.6549
##     38     0.8234  0.323965  0.6556
##     39     0.8230  0.324591  0.6551
##     40     0.8228  0.325095  0.6549
##     41     0.8207  0.328268  0.6536
##     42     0.8202  0.329158  0.6546
##     43     0.8189  0.331183  0.6534
##     44     0.8180  0.332494  0.6524
##     45     0.8174  0.333279  0.6518
##     46     0.8185  0.331533  0.6527
##     47     0.8189  0.331030  0.6531
##     48     0.8192  0.330613  0.6525
##     49     0.8201  0.329215  0.6538
##     50     0.8209  0.327942  0.6547
##     51     0.8214  0.327226  0.6551
##     52     0.8203  0.328965  0.6546
##     53     0.8218  0.326569  0.6559
##     54     0.8216  0.326928  0.6558
##     55     0.8216  0.326885  0.6561
##     56     0.8219  0.326390  0.6568
##     57     0.8216  0.326907  0.6567
##     58     0.8220  0.326643  0.6564
##     59     0.8223  0.326078  0.6568
##     60     0.8230  0.325160  0.6570
##     61     0.8229  0.325431  0.6574
##     62     0.8231  0.325419  0.6574
##     63     0.8230  0.325680  0.6573
##     64     0.8227  0.326160  0.6566
##     65     0.8239  0.324183  0.6576
##     66     0.8243  0.323588  0.6582
##     67     0.8244  0.323544  0.6582
##     68     0.8251  0.322403  0.6587
##     69     0.8253  0.322149  0.6586
##     70     0.8260  0.321095  0.6596
##     71     0.8264  0.320646  0.6595
##     72     0.8273  0.319352  0.6603
##     73     0.8281  0.318152  0.6609
##     74     0.8287  0.317210  0.6616
##     75     0.8282  0.318031  0.6608
##     76     0.8290  0.317003  0.6619
##     77     0.8285  0.317813  0.6611
##     78     0.8290  0.317286  0.6610
##     79     0.8294  0.316716  0.6611
##     80     0.8307  0.314846  0.6623
##     81     0.8255  0.323601  0.6566
##     82     0.8258  0.323011  0.6566
##     83     0.8268  0.321601  0.6573
##     84     0.8278  0.320243  0.6580
##     85     0.8286  0.319024  0.6590
##     86     0.8282  0.319643  0.6588
```

```
##    87    0.8269  0.321770  0.6576
##    88    0.8280  0.320430  0.6582
##    89    0.8246  0.326539  0.6532
##    90    0.8232  0.328920  0.6525
##    91    0.8252  0.326531  0.6544
##    92    0.8244  0.328011  0.6543
##    93    0.8245  0.328031  0.6547
##    94    0.8248  0.327608  0.6550
##    95    0.8255  0.326714  0.6555
##    96    0.8262  0.325647  0.6559
##    97    0.8270  0.324881  0.6561
##    98    0.8269  0.325059  0.6557
##    99    0.8272  0.324811  0.6555
##   100    0.8281  0.323421  0.6569
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 45.
```

```r
# extract best ncomp
myGrid <- expand.grid(ncomp = gss_pcr_caret$bestTune$ncomp)

# fit model with best ncomp
gss_pcr_best = train(
  egalit_scale ~ ., data = gss_test,
  method = "pcr",
  tuneGrid = myGrid
)
gss_pcr_best
```

```
## Principal Component Analysis
##
## 493 samples
##  44 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 493, 493, 493, 493, 493, 493, ...
## Resampling results:
##
##   RMSE    Rsquared  MAE
##   0.8845  0.2537    0.7093
##
## Tuning parameter 'ncomp' was held constant at a value of 45
```

After tuning, the best number of principal components is 45. The train CV MSE is 0.6681 and the final MSE is 0.7823. The final MSE is worse than that of elastic net but better than that of linear regression. It is likely worse than elastic net because PCA is not suitable for categorical variables. However, it is still better than linear regression likely because there are some shared variance among the predictors.

d) PLS

```r
# find best ncomp
gss_pls_caret = train(
  egalit_scale ~ ., data = gss_train,
```

```r
  method = "pls",
  trControl = cv_10,
  tuneLength = 100
)
gss_pls_caret
```

```
## Partial Least Squares
##
## 1481 samples
##   44 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1333, 1333, 1333, 1332, 1332, 1334, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE    Rsquared  MAE
##     1    0.8766  0.2331    0.7125
##     2    0.8382  0.3025    0.6678
##     3    0.8283  0.3187    0.6586
##     4    0.8198  0.3329    0.6485
##     5    0.8159  0.3402    0.6469
##     6    0.8181  0.3372    0.6486
##     7    0.8196  0.3352    0.6494
##     8    0.8219  0.3324    0.6515
##     9    0.8230  0.3311    0.6515
##    10    0.8234  0.3308    0.6520
##    11    0.8232  0.3315    0.6524
##    12    0.8222  0.3335    0.6500
##    13    0.8211  0.3351    0.6496
##    14    0.8207  0.3354    0.6486
##    15    0.8199  0.3366    0.6482
##    16    0.8200  0.3364    0.6476
##    17    0.8201  0.3361    0.6480
##    18    0.8202  0.3359    0.6485
##    19    0.8209  0.3349    0.6487
##    20    0.8208  0.3351    0.6486
##    21    0.8213  0.3343    0.6491
##    22    0.8216  0.3339    0.6492
##    23    0.8218  0.3336    0.6496
##    24    0.8218  0.3336    0.6497
##    25    0.8219  0.3335    0.6495
##    26    0.8222  0.3333    0.6498
##    27    0.8224  0.3329    0.6499
##    28    0.8226  0.3327    0.6500
##    29    0.8231  0.3320    0.6504
##    30    0.8235  0.3315    0.6506
##    31    0.8242  0.3306    0.6511
##    32    0.8246  0.3301    0.6514
##    33    0.8248  0.3298    0.6515
##    34    0.8253  0.3293    0.6517
##    35    0.8256  0.3289    0.6519
##    36    0.8260  0.3285    0.6521
##    37    0.8261  0.3283    0.6522
```

```
##   38   0.8263 0.3281   0.6523
##   39   0.8266 0.3278   0.6525
##   40   0.8267 0.3276   0.6526
##   41   0.8268 0.3275   0.6527
##   42   0.8270 0.3273   0.6528
##   43   0.8269 0.3274   0.6527
##   44   0.8269 0.3274   0.6526
##   45   0.8269 0.3273   0.6526
##   46   0.8270 0.3272   0.6527
##   47   0.8271 0.3271   0.6528
##   48   0.8271 0.3270   0.6528
##   49   0.8272 0.3269   0.6528
##   50   0.8273 0.3268   0.6528
##   51   0.8273 0.3269   0.6528
##   52   0.8273 0.3268   0.6528
##   53   0.8273 0.3268   0.6528
##   54   0.8273 0.3268   0.6528
##   55   0.8273 0.3268   0.6527
##   56   0.8273 0.3268   0.6528
##   57   0.8273 0.3268   0.6528
##   58   0.8273 0.3268   0.6528
##   59   0.8273 0.3268   0.6528
##   60   0.8273 0.3268   0.6527
##   61   0.8273 0.3268   0.6527
##   62   0.8273 0.3268   0.6528
##   63   0.8273 0.3268   0.6528
##   64   0.8273 0.3268   0.6528
##   65   0.8273 0.3268   0.6528
##   66   0.8273 0.3268   0.6528
##   67   0.8273 0.3268   0.6528
##   68   0.8273 0.3268   0.6528
##   69   0.8273 0.3268   0.6528
##   70   0.8273 0.3268   0.6528
##   71   0.8273 0.3268   0.6528
##   72   0.8273 0.3268   0.6528
##   73   0.8273 0.3268   0.6528
##   74   0.8273 0.3268   0.6528
##   75   0.8273 0.3268   0.6528
##   76   0.8273 0.3268   0.6528
##   77   0.8273 0.3268   0.6528
##   78   0.8273 0.3268   0.6528
##   79   0.8273 0.3268   0.6528
##   80   0.8273 0.3268   0.6528
##   81   0.8273 0.3268   0.6528
##   82   0.8273 0.3268   0.6528
##   83   0.8273 0.3268   0.6528
##   84   0.8273 0.3268   0.6528
##   85   0.8273 0.3268   0.6528
##   86   0.8273 0.3268   0.6528
##   87   0.8273 0.3268   0.6528
##   88   0.8273 0.3268   0.6528
##   89   0.8273 0.3268   0.6528
##   90   0.8273 0.3268   0.6528
##   91   0.8273 0.3268   0.6528
```

```
##    92    0.8273  0.3268    0.6528
##    93    0.8273  0.3268    0.6528
##    94    0.8273  0.3268    0.6528
##    95    0.8273  0.3268    0.6528
##    96    0.8273  0.3268    0.6528
##    97    0.8273  0.3268    0.6528
##    98    0.8273  0.3268    0.6528
##    99    0.8273  0.3268    0.6528
##   100    0.8273  0.3268    0.6528
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 5.
```

```r
# extract best ncomp
myGrid <- expand.grid(ncomp = gss_pls_caret$bestTune$ncomp)

# fit model with best ncomp
gss_pls_best = train(
  egalit_scale ~ ., data = gss_test,
  method = "pls",
  tuneGrid = myGrid
)
gss_pls_best
```

```
## Partial Least Squares
##
## 493 samples
##  44 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 493, 493, 493, 493, 493, 493, ...
## Resampling results:
##
##   RMSE   Rsquared  MAE
##   0.905  0.2316    0.7226
##
## Tuning parameter 'ncomp' was held constant at a value of 5
```

After tuning, the best number of principal components is 5, significantly lower than PCR. This is expected because PLS takes `egalit_scale` into consideration and hence can reach the lowest train CV MSE before PCR. The train CV MSE is 0.6657 and the final MSE is 0.819. Its final MSE is worse than PCR's likely because its optimal number of components is way lower than PCR's. The same concern about using PCA with categorical variables as in PCR still remains. It, nevertheless, still has better final MSE than linear regression.

Quick Summary: Elastic net seems to be the best model based on MSE. PCR and PLS probably don't do as well because of the categorical variables. Still, PCR and PLS do better than Linear Regression.

5) Egalitarianism and everything (Model Agnostic)

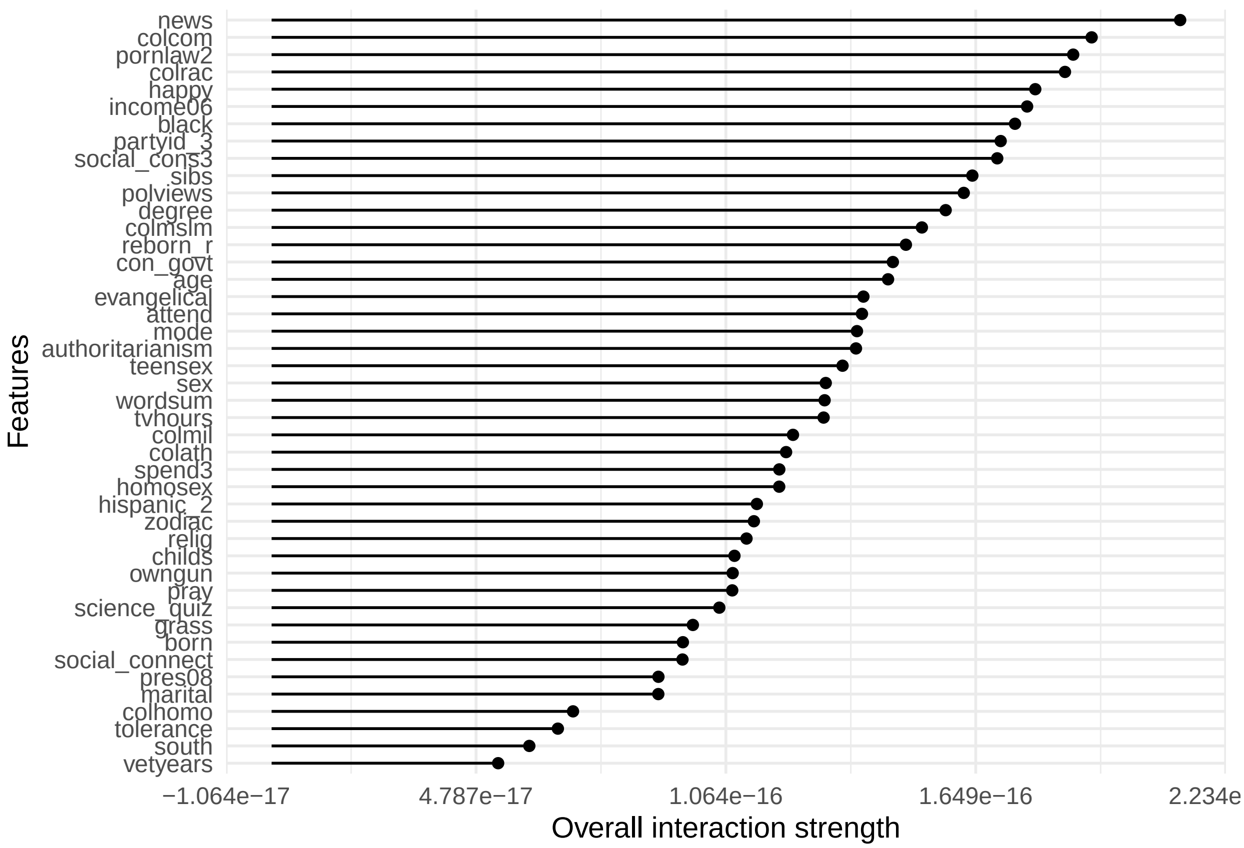
```r
# X and y
features <- gss_test %>%
```

```r
  dplyr::select(-egalit_scale)
response <- as.numeric(as.vector(gss_test$egalit_scale))
```

a) Linear Regression

```r
predictor.lm_caret <- iml::Predictor$new(
  model = lm_caret,
  data = features,
  y = response)

# interaction plot
lm_caret.inx <- Interaction$new(predictor.lm_caret)
plot(lm_caret.inx)
```



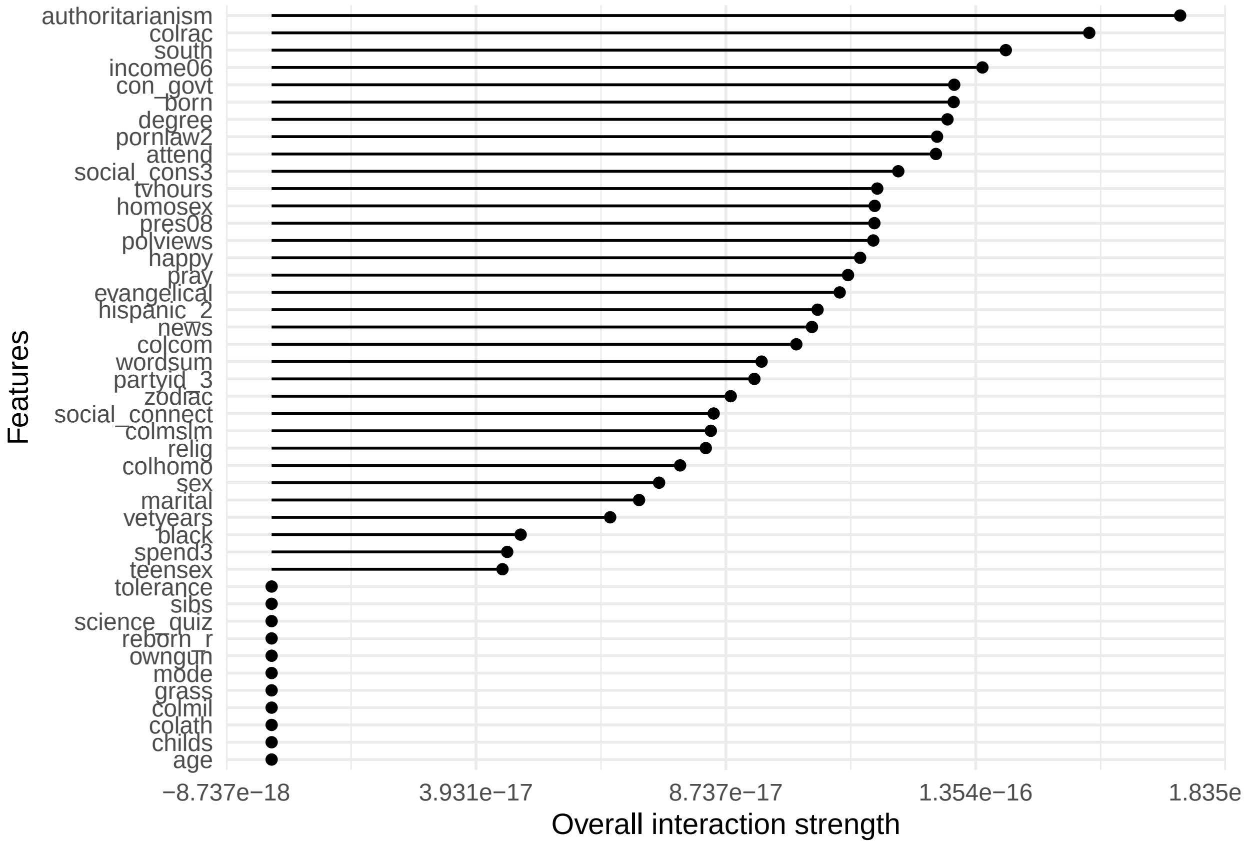
`news` has the strongest interaction strength in linear regression, followed by `colcom` and `pornlaw2`.

b) Elastic Net Regression

```r
# predict function for elastic model
predict_glmnet <- function(model, newdata){
  predict(model, s='lambda.min',
          newx=model.matrix(egalit_scale ~., data = gss_train)[, -1])
}
```

```r
predictor.elnet <- iml::Predictor$new(
  model = gss_elnet_best,
  data = features,
  y = response,
  predict.fun = predict_glmnet)

# interaction plot
elas.inx <- Interaction$new(predictor.elnet)
plot(elas.inx)
```



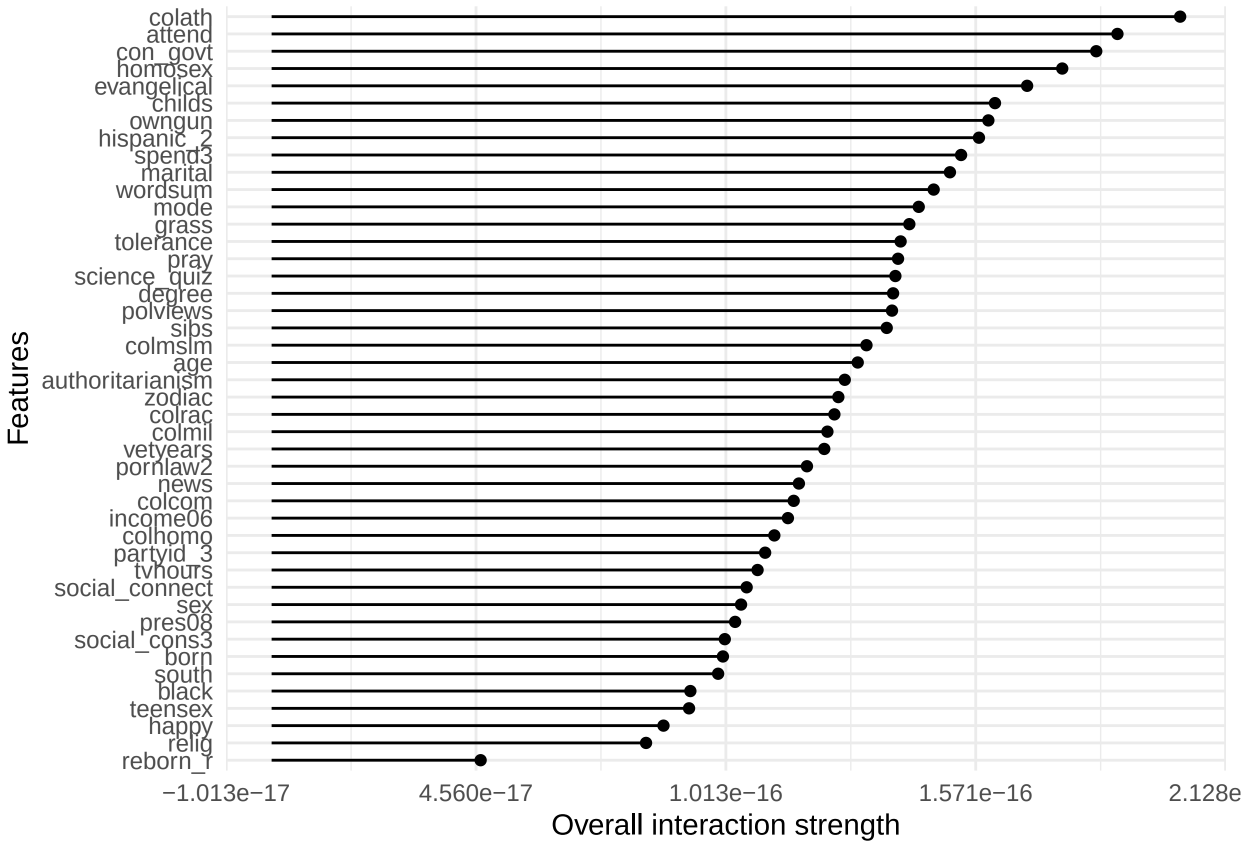
`authoritarianism` has the strongest interaction strength in elastic net, followed by `colrac` and `south`. Note that features' interaction strengths are weaker in comparison to those in linear regression (note difference in x-axis scales). This could be because elastic net penalize very big coefficients, that is penalizing some features being too influencial. In addition, there is a huge drop in interaction strength starting from `tolerence`. This likely reflects the ridge part of elastic net eliminates some of the features.

c) PCR

```r
predictor.pcr <- iml::Predictor$new(
  model = gss_pcr_best,
  data = features,
  y = response)

# interaction plot
pcr.inx <- Interaction$new(predictor.pcr)
```

```r
plot(pcr.inx)
```

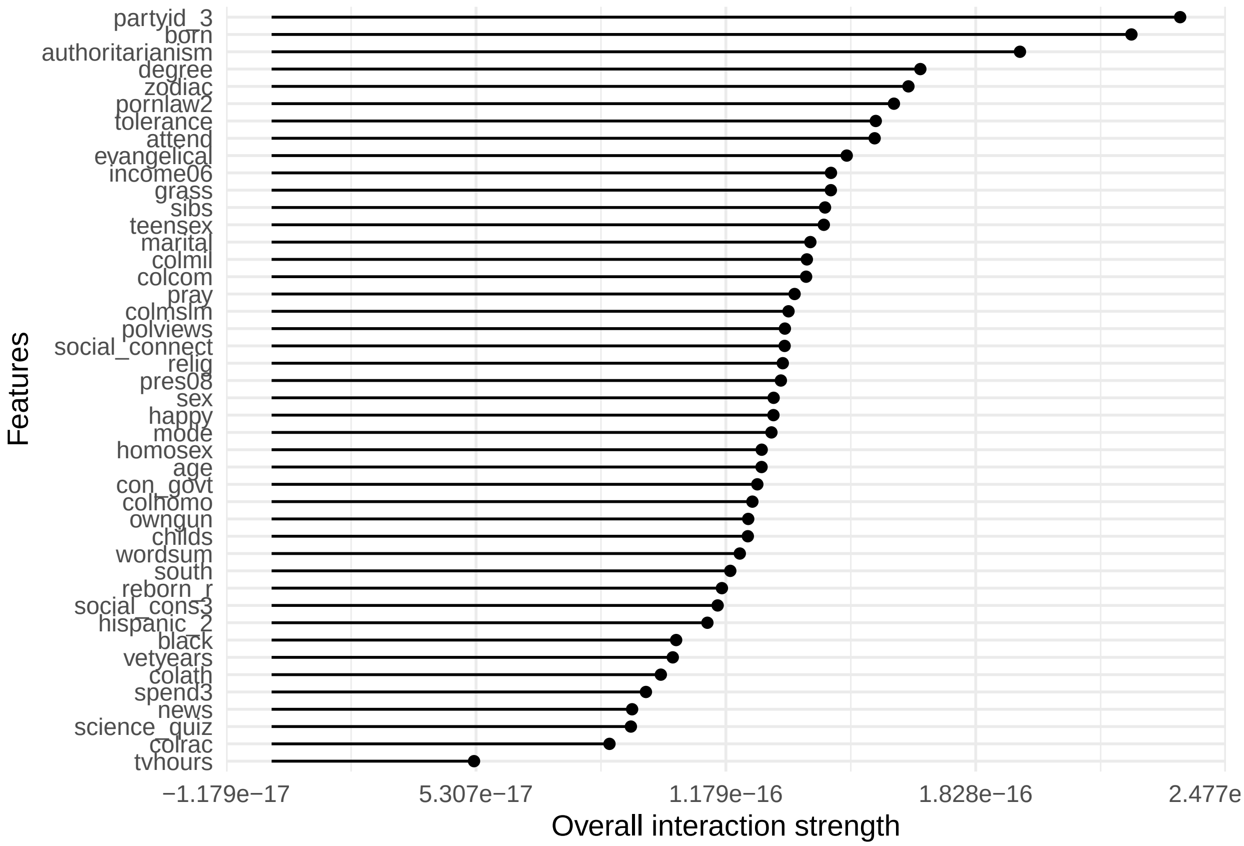


`colath` has the strongest interaction strength in PCR, followed by `attend` and `con_govt`.

d) PLS

```r
predictor.pls <- iml::Predictor$new(
  model = gss_pls_best,
  data = features,
  y = response)

# interaction plot
pls.inx <- Interaction$new(predictor.pls)
plot(pls.inx)
```



`partyid_3` has the strongest interaction strength in PLS, followed by `born` and `authoritarianism`. The top 3 features are visibly more prominant than other features, unlike PCR's where the differences are more subtle. This is likely because its optimal number of components is way lower than PCR's.

Models Comparison

```
top_lm_inx <- arrange(lm_caret.inx$results, desc(.interaction))[1:15,1]
top_elas_inx <- arrange(elas.inx$results, desc(.interaction))[1:15,1]
top_pcr_inx <- arrange(pcr.inx$results, desc(.interaction))[1:15,1]
top_pls_inx <- arrange(pls.inx$results, desc(.interaction))[1:15,1]
data_frame("Rank" = c(1:15), "LM" = top_lm_inx,
           "Elastic Net" = top_elas_inx,
           "PCR" = top_pcr_inx, "PLS" = top_pls_inx) %>%
  kable(caption = "Top 15 Important features by Models")
```

Table 4: Top 15 Important features by Models

Rank	LM	Elastic Net	PCR	PLS
1	news	authoritarianism	colath	partyid_3
2	colcom	colrac	attend	born
3	pornlaw2	south	con_govt	authoritarianism
4	colrac	income06	homosex	degree
5	happy	con_govt	evangelical	zodiac
6	income06	born	childs	pornlaw2
7	black	degree	owngun	tolerance

Rank	LM	Elastic Net	PCR	PLS
8	partyid_3	pornlaw2	hispanic_2	attend
9	social_cons3	attend	spend3	evangelical
10	sibs	social_cons3	marital	income06
11	polviews	tvhours	wordsum	grass
12	degree	homosex	mode	sibs
13	colmslm	pres08	grass	teensex
14	reborn_r	polviews	tolerance	marital
15	con_govt	happy	pray	colmil

All models have different rank of important features by interaction strength. This is expected because each model prioritizes and penalizes different things. Some features that seem to be in top 15 across the majority of the models are: `con_govt`, `pornlaw2`, `income06`, `degree`, `attend`. Still, these features are ranked very differently across models.

Let's look at `income06` and `degree` in across models:

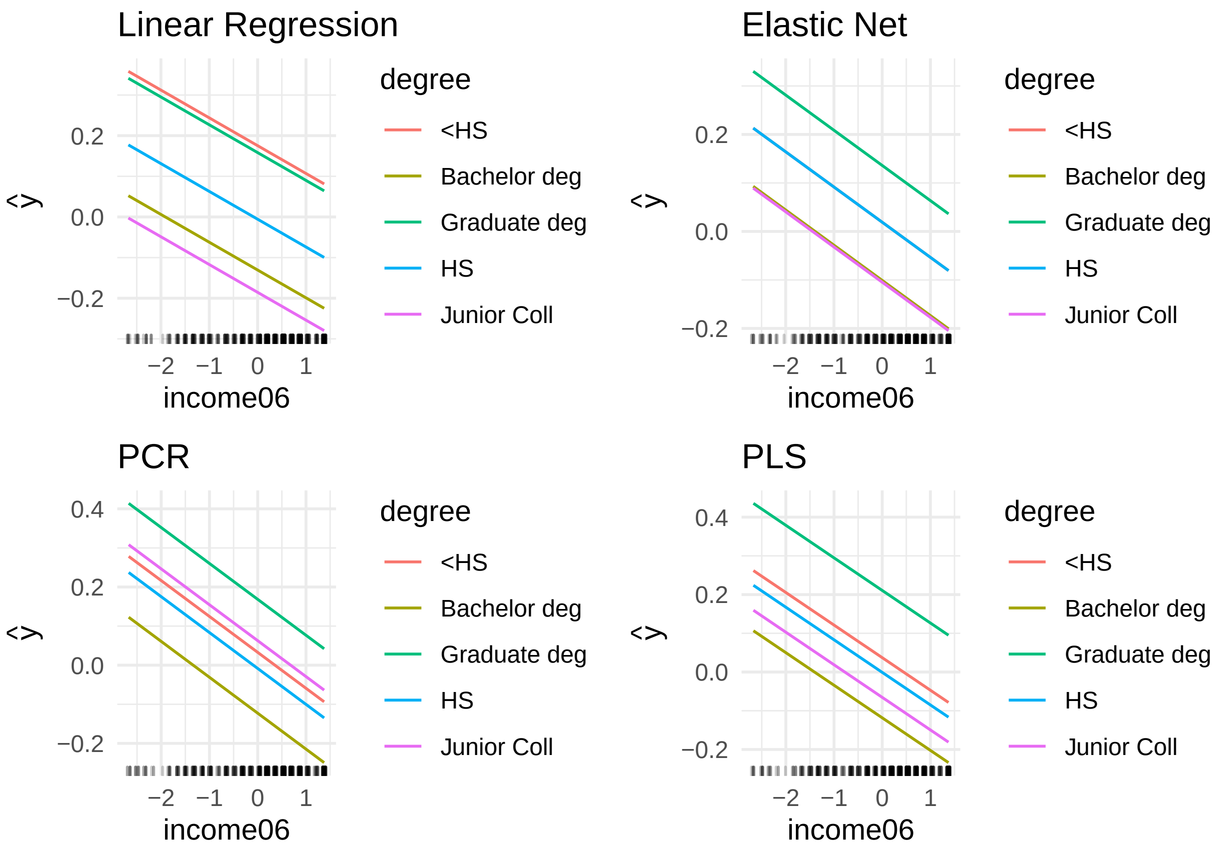
```
lm.income06 <- Partial$new(predictor.lm_caret, c("income06", "degree"), grid.size = 50)
p1 <- plot(lm.income06) + ggtitle("Linear Regression")

elnet.income06 <- Partial$new(predictor.elnet, c("income06", "degree"), grid.size = 50)
p2 <- plot(elnet.income06) + ggtitle("Elastic Net")

pcr.income06 <- Partial$new(predictor.pcr, c("income06", "degree"), grid.size = 50)
p3 <- plot(pcr.income06) + ggtitle("PCR")

pls.income06 <- Partial$new(predictor.pls, c("income06", "degree"), grid.size = 50)
p4 <- plot(pls.income06) + ggtitle("PLS")

gridExtra::grid.arrange(p1, p2, p3, p4, nrow = 2)
```



From the plot, we can see that `income06` and `degree` affect predicted `egalit_scale` differently in different models. For example, even though the effect of `income06` is linear in all models, slope of `income06` in PCR and PLS is more negative than that in linear model and elastic net; `degree:<HS` has highest predicted `egalit_scale` in linear regression but not in the other three models; etc.

Quick Summary: Even though all 4 models use linear approach, they are still very different because they prioritize and penalize different things. We should choose appropriate model considering model's accuracy as well as the goal of the task for the model