Final Report of Group 26: Supervised Statistical Learning with 2016 NFL Data Table Fields

Contents

Names of students

Name	NetID
Yuming Zhang	yzhan216
Rex Zhou	rzhou12
Wei Zhang	wzhng100

Set up

```
# load necessary packages
library(readr)
library(caret)
library(MASS)
library(gam)
library(gbm)
library(rpart)
library(randomForest)
library(glmnet)
library(plyr)
#library(corrplot)
library(lattice)
library(rpart.plot)
library(klaR)
```

```r
# define functions needed
get_best_result = function(caret_fit) {
  best_result = caret_fit$results[as.numeric(rownames(caret_fit$bestTune)), ]
  rownames(best_result) = NULL
  best_result
}
```

Introduction

In this final project, we are going to accomplish the statistical learning task of regression with the dataset called `New_Team.csv`. This dataset includes the performance of each NFL team in 2016. The data comes from Armchair Analysis (http://armchairanalysis.com/), which is a service that provides affordable NFL data for the past 15 years. As our task is regression, we will use `RMSE` as our metric throughout the whole data analysis, which is defined below.

```r
# define "rmse" function
rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}
```

Firstly, let's read in the `New_Team.csv` dataset and have a quick look at how this dataset is like.

```r
# read in data
team_data = read_csv("New_Team.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   tname = col_character(),
##   lnr = col_double(),
##   lnp = col_double(),
##   lbr = col_double(),
##   lbp = col_double(),
##   dbr = col_double(),
##   dbp = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

```r
# quick look at the dataset
dim(team_data)
```

```
## [1] 8512   26
```

```r
head(team_data, 10)
```

```
## # A tibble: 10 × 26
##      gid tname   pts    pc   fum    ir    td   fgm   srp   spp   spc   mpc
##    <int> <chr> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
## 1      1    SF    28    23     0     0     4     0    10    20     3    15
## 2      1   ATL    36    16     1     1     3     5    11    14     2     6
## 3      2   JAC    27    24     0     0     3     2    19    23     3    14
## 4      2   CLE     7    19     1     0     1     0     9    12    10     5
## 5      3   PHI    41    16     0     2     5     2    28    12     6     8
## 6      3   DAL    14    13     0     3     1     2     4     8     5     5
```

```
## 7      4   NYJ    20    23     0     1     2     2    13    21     7    11
## 8      4    GB    16    14     1     1     1     3     8    12     5     8
## 9      5   IND    27    22     1     1     3     2    15    18     7     7
## 10     5    KC    14    16     0     1     2     0     8    13     3     8
## # ... with 14 more variables: lpc <int>, sky <int>, sfpy <int>, drv <int>,
## #   npy <int>, lnr <dbl>, lnp <dbl>, lbr <dbl>, lbp <dbl>, dbr <dbl>,
## #   dbp <dbl>, stf <int>, ohp <int>, pfn <int>
```

As we can see, the dataset has 8512 observation and 26 variables. Here we list some variables that are of greatest importance for our data analysis later.

Variable	Explanation
pts	Points Scored
td	Touchdowns
fgm	Field Goals Made
spp	Successful Pass Plays
pc	Completions

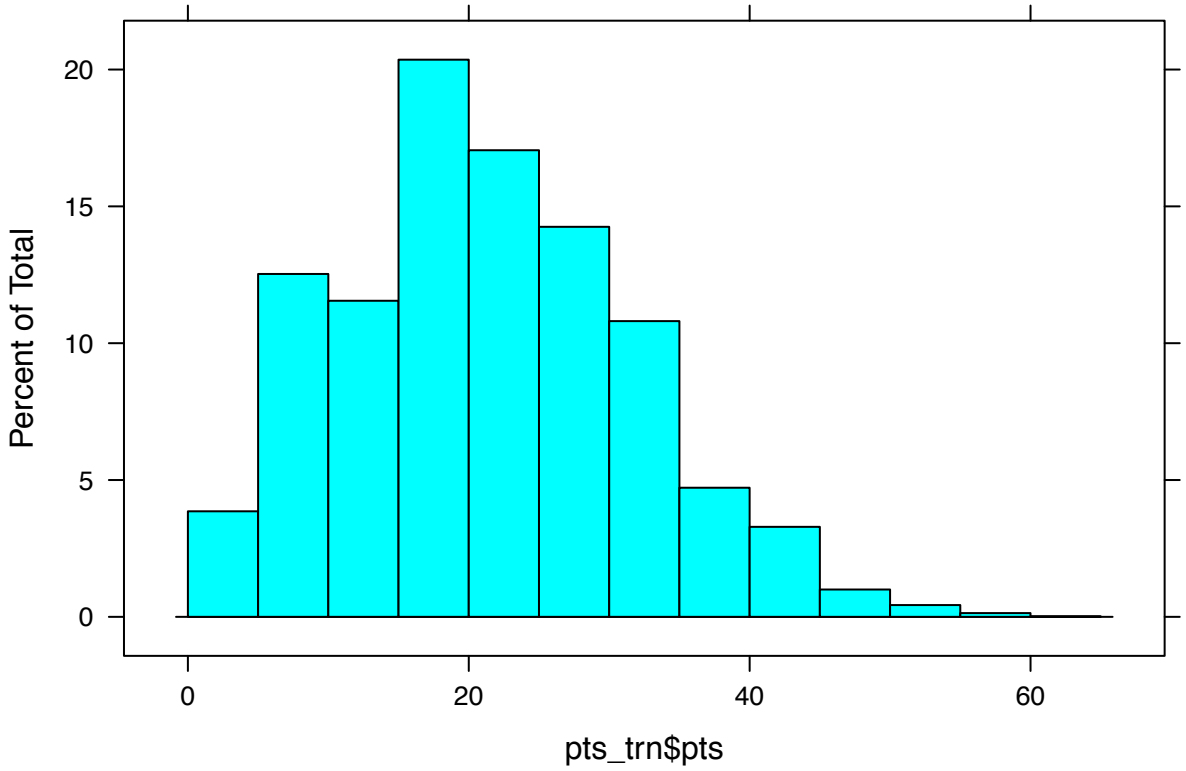
For our following data analysis, we will use `pts` as the response variable and select some of the remaining 25 variables as predictors for our methods.

Secondly, we perform the test-train split on the dataset. We will use 60% of the data as our train dataset, and the remaining 40% as our test dataset.

```
# test-train split
set.seed(26)
pts_idx = createDataPartition(team_data$pts, p = 0.6, list = FALSE)
pts_trn = team_data[pts_idx,]
pts_tst = team_data[-pts_idx,]
```

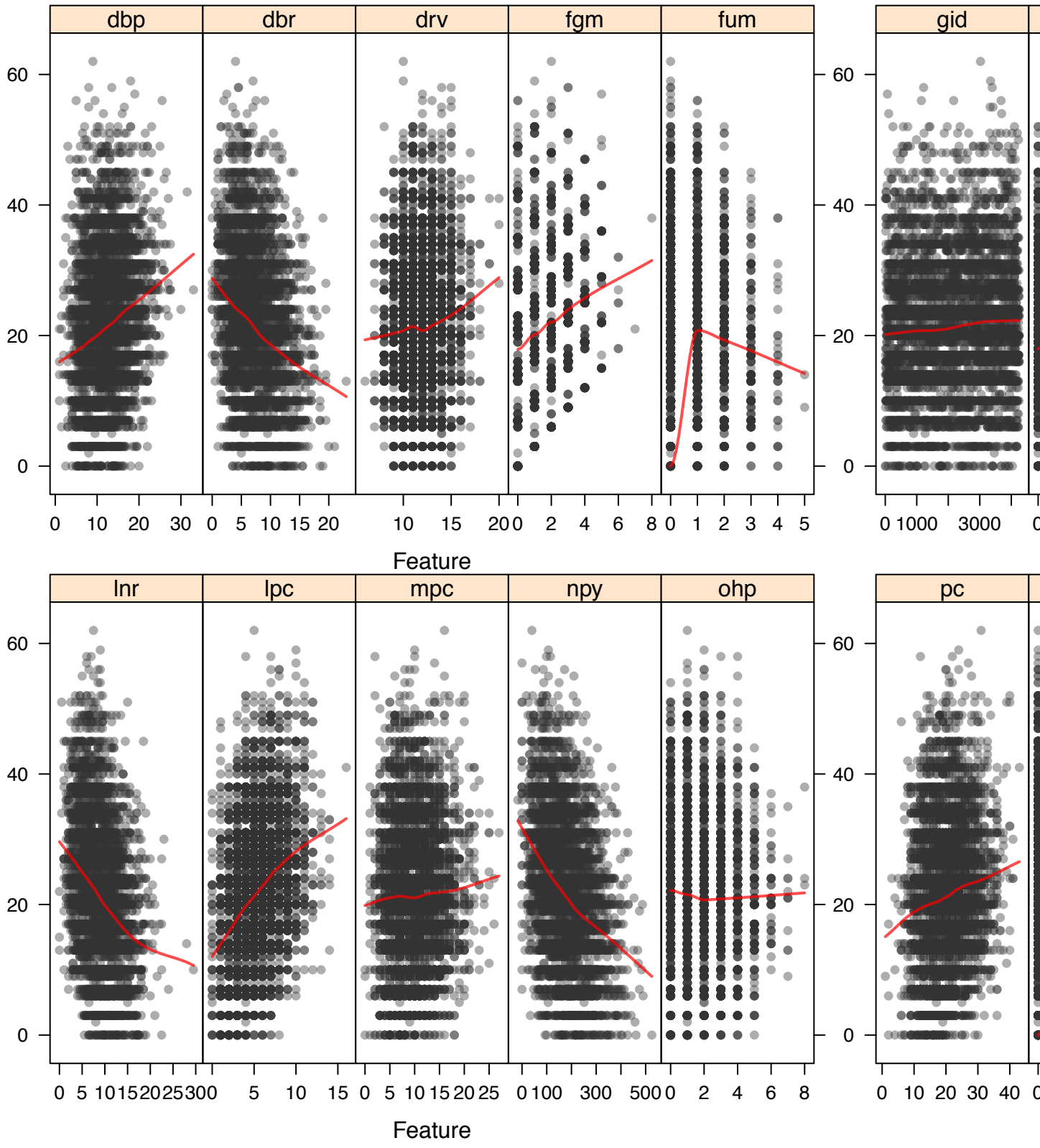
In order to have a better understanding of the dataset and to find the most significant subset of predictors, we perform some visualization of the dataset.

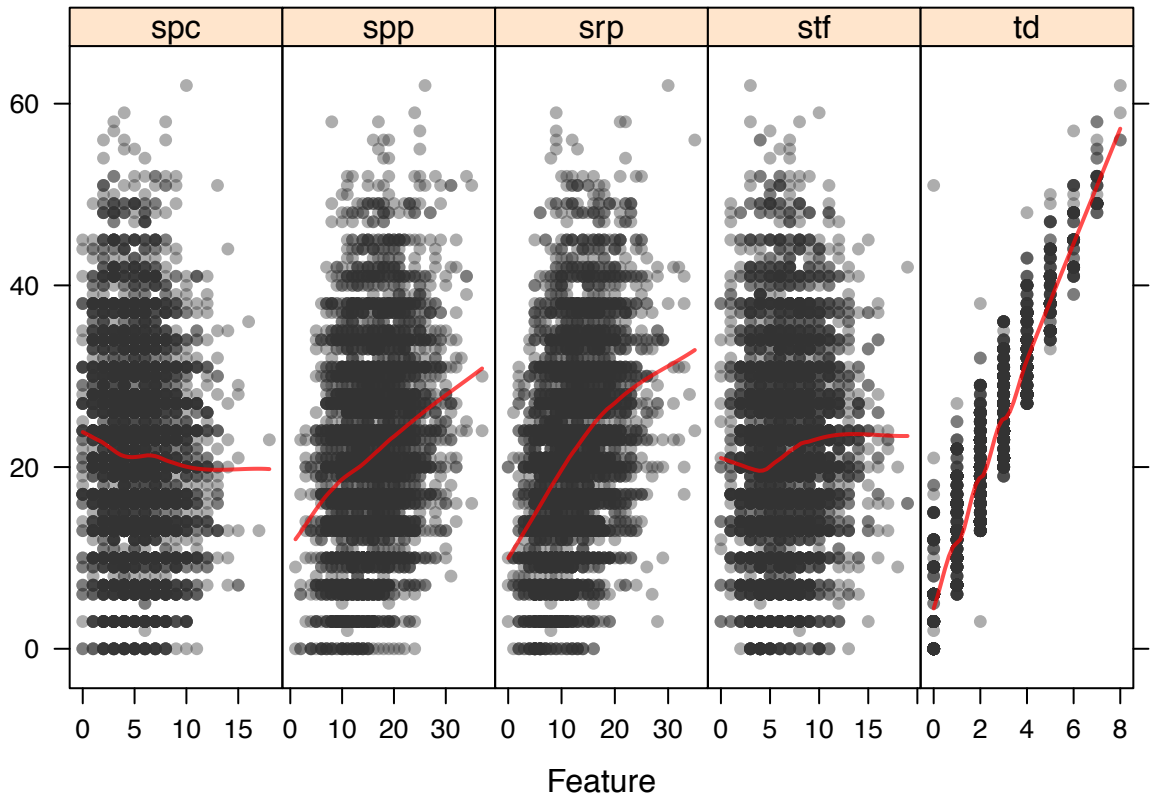
```
histogram(pts_trn$pts, breaks = 20)
```



The following plots will explore the relationship of the feature variables with the response variable.

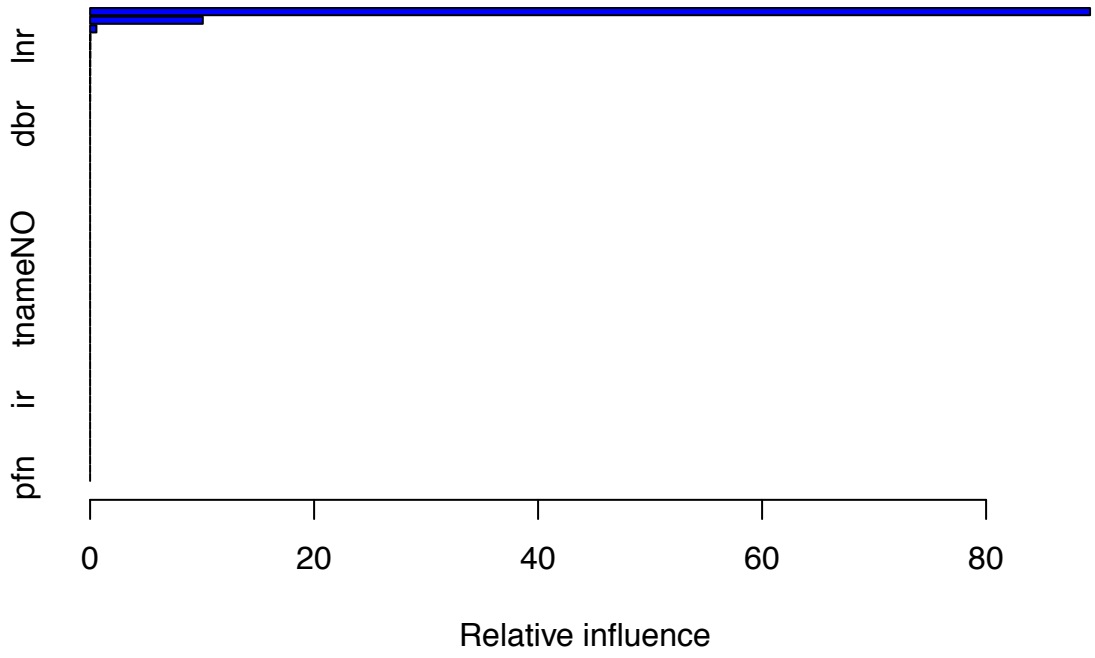
```
theme1 = trellis.par.get()
theme1$plot.symbol$col = rgb(.2, .2, .2, .4)
theme1$plot.symbol$pch = 16
theme1$plot.line$col = rgb(1, 0, 0, .7)
theme1$plot.line$lwd = 2

trellis.par.set(theme1)

featurePlot(x = pts_trn[, -2],
            y = pts_trn$pts,
            plot = "scatter",
            type = c("p", "smooth"),
            span = .5,
            layout = c(5, 1))
```


dbp
dbr
drv
fgm
fum
gid
lnr
lpc
mpc
npy
ohp
pc
Feature
Feature



From the above plot, we can find many potential predictors, including `dbp`, `dbr`, `drv`, `fgm`, `ir`, `lbp`, `lbr`, `lnr`, `lpc`, `npy`, `pc`, `sfpy`, `sky`, `spp`, `srp`, `td`. So we will need to fit a boosted tree model to further narrow down our subset of predictors.

```
cv_5 = trainControl(method = "cv", number = 5)
gbm_grid = expand.grid(interaction.depth = c(1, 2),
                       n.trees = c(500, 1000, 1500),
                       shrinkage = c(0.001, 0.01, 0.1),
                       n.minobsinnode = 10)
gbm_tune = train(pts ~ ., data = pts_trn,
                 method = "gbm",
                 trControl = cv_5,
                 verbose = FALSE,
                 tuneGrid = gbm_grid,
                 preProcess = c("scale"))
result = summary(gbm_tune)
```




```
result[1:5, ]
```

```
##     var     rel.inf
## td   td 89.28248186
## fgm fgm 10.06223752
## spp spp  0.56501820
## pc   pc  0.02804917
## lnr lnr  0.01810702
```

The above information shows that `td`, `fgm`, `spp` and `pc` are the best predictors. We will consider these 4 predictors for the following methods.

Methods

For our data analysis, we will consider the following methods: - Linear regression - Penalized linear regression (Elastic net) - Trees (Random forest) - Generalized Additive Models (GAMs)

For each method we will consider different sets of features: - `small`: only `td`, `fgm`, `spp` and `pc` - `int`: significant interaction between `td`, `fgm`, `spp` and `pc`. That is, `td + fgm + spp + pc + td:spp + fgm:spp + td:pc + fgm:pc`. - `full`: all features - `huge`: all features with all possible two way interactions

In particular, for `int`, we fit the linear regression model with all interactions between `td`, `fgm`, `spp` and `pc` to obtain the significant interactions.

```
mod = train(pts ~ td * fgm * spp * pc, data = pts_trn, method = "lm", trControl = cv_5)
summary(mod)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -12.657  -0.148  -0.039   0.066  45.863
```

```
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.0757867  0.2861670   0.265 0.791148
## td                7.0894471  0.1034198  68.550  < 2e-16 ***
## fgm               3.0436671  0.1567272  19.420  < 2e-16 ***
## spp               0.2197286  0.0271991   8.079 8.12e-16 ***
## pc               -0.1470896  0.0217141  -6.774 1.40e-11 ***
## `td:fgm`         -0.0353401  0.0624410  -0.566 0.571436
## `td:spp`         -0.0550389  0.0089710  -6.135 9.15e-10 ***
## `fgm:spp`        -0.0632948  0.0141902  -4.460 8.35e-06 ***
## `td:pc`           0.0276015  0.0076186   3.623 0.000294 ***
## `fgm:pc`          0.0370964  0.0112138   3.308 0.000946 ***
## `spp:pc`         -0.0011587  0.0009228  -1.256 0.209271
## `td:fgm:spp`      0.0198918  0.0052077   3.820 0.000135 ***
## `td:fgm:pc`      -0.0102776  0.0043603  -2.357 0.018455 *
## `td:spp:pc`       0.0005368  0.0002833   1.895 0.058153 .
## `fgm:spp:pc`      0.0005541  0.0004841   1.145 0.252436
## `td:fgm:spp:pc` -0.0002109  0.0001665  -1.267 0.205143
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9069 on 5092 degrees of freedom
## Multiple R-squared:  0.9924, Adjusted R-squared:  0.9923
## F-statistic: 4.414e+04 on 15 and 5092 DF,  p-value: < 2.2e-16
```

So we select the following significant interactions: `td:spp`, `fgm:spp`, `td:pc`, `fgm:pc`.

Resampling

Throughout the whole data analysis, we will use 5-fold cross-validation resampling method to tune the best model for each method and each set of features.

```
cv_5 = trainControl(method = "cv", number = 5, verbose = FALSE)
```

Linear regression

Linear regression is a linear, parametric, generative method. By 5-fold cross-validation, we can tune the parameters of intercept and all coefficients to reduce the randomness and find the best fitted model.

```
# lm + small
lm_small = train(pts ~ td + fgm + spp + pc,
                 data = pts_trn, method = "lm", trControl = cv_5)

# lm + int
lm_int = train(pts ~ td + fgm + spp + pc + td:spp + fgm:spp + td:pc + fgm:pc,
               data = pts_trn, method = "lm", trControl = cv_5)

# lm + full
lm_full = train(pts ~ ., data = pts_trn, method = "lm", trControl = cv_5)

# cv train rmse
lm_small_trn_rmse = get_best_result(lm_small)$RMSE
```

```
lm_int_trn_rmse = get_best_result(lm_int)$RMSE
lm_full_trn_rmse = get_best_result(lm_full)$RMSE

# test rmse
lm_small_tst_rmse = rmse(actual = pts_tst$pts, predicted = predict(lm_small, pts_tst))
lm_int_tst_rmse = rmse(actual = pts_tst$pts, predicted = predict(lm_int, pts_tst))
lm_full_tst_rmse = rmse(actual = pts_tst$pts, predicted = predict(lm_full, pts_tst))
```

Penalized linear regression (Elastic net)

Penalized linear regression is a linear, parametric, generative method with shrinkage method elastic net that combines the ridge and lasso methods. The tuning parameters for this method are `alpha` and `lambda`.

```
# glmnet + small
glmn_small = train(pts ~ td + fgm + spp + pc,
                   data = pts_trn[, -c(1,2)],
                   method = "glmnet",
                   trControl = cv_5,
                   tuneLength = 10)

# glmnet + int
glmn_int   = train(pts ~ td + fgm + spp + pc + td:spp + fgm:spp + td:pc + fgm:pc,
                   data = pts_trn[, -c(1, 2)],
                   method = "glmnet",
                   trControl = cv_5,
                   tuneLength = 10)

# glmnet + full
glmn_full = train(pts ~ ., data = pts_trn[, -c(1,2)], method = "glmnet",
                   trControl = cv_5, tuneLength = 10)

# glmnet + huge
glmn_huge  = train(pts ~ . ^ 2, data = pts_trn[, -c(1,2)], method = "glmnet",
                   trControl = cv_5, tuneLength = 10)

# cv train rmse
glmn_small_trn_rmse = get_best_result(glmn_small)$RMSE
glmn_int_trn_rmse = get_best_result(glmn_int)$RMSE
glmn_full_trn_rmse = get_best_result(glmn_full)$RMSE
glmn_huge_trn_rmse = get_best_result(glmn_huge)$RMSE

# test rmse
glmn_small_tst_rmse = rmse(pts_tst$pts, predict(glmn_small, pts_tst))
glmn_int_tst_rmse = rmse(pts_tst$pts, predict(glmn_int, pts_tst))
glmn_full_tst_rmse = rmse(pts_tst$pts, predict(glmn_full, pts_tst))
glmn_huge_tst_rmse = rmse(pts_tst$pts, predict(glmn_huge, pts_tst))
```

Trees (Random forest)

Tree is a non-linear, non-parametric, discriminative method. There are some ensemble methods of trees including bagging, random forest and boosting. Here we use the ensemble method of random forest. The tuning parameter is `mtry` in this case.

```r
# random forest grid
rf_grid = expand.grid(mtry = c(1, 2, 3, 4))

# rf + small
rf_small = train(pts ~ td + fgm + spp + pc,
                 data = pts_trn, trControl = cv_5,
                 method = "rf", tuneGrid = rf_grid)

# rf + int
rf_int = train(pts ~ td + fgm + spp + pc + td:spp + fgm:spp + td:pc + fgm:pc,
               data = pts_trn, trControl = cv_5, method = "rf", tuneGrid = rf_grid)

# cv train rmse
rf_small_trn_rmse = get_best_result(rf_small)$RMSE
rf_int_trn_rmse = get_best_result(rf_int)$RMSE

# test rmse
rf_small_tst_rmse = rmse(pts_tst$pts, predict(rf_small, pts_tst))
rf_int_tst_rmse = rmse(pts_tst$pts, predict(rf_int, pts_tst))
```

Generalized Additive Models (GAMs)

GAMs is a linear, parametric, generative method. The tuning parameter is `degrees of freedom (df)`.

```r
# GAM grid
gam_grid = expand.grid(df = 1:10)

# GAM + small
gam_small = train(pts ~ td + fgm + spp + pc,
                  data = pts_trn, trControl = cv_5,
                  method = "gamSpline", tuneGrid = gam_grid)

# GAM + full
gam_full = train(pts ~ ., data = pts_trn, trControl = cv_5,
                  method = "gamSpline", tuneGrid = gam_grid)

# cv train rmse
gam_small_trn_rmse = get_best_result(gam_small)$RMSE
gam_full_trn_rmse = get_best_result(gam_full)$RMSE

# test rmse
gam_small_tst_rmse = rmse(pts_tst$pts, predict(gam_small, pts_tst))
gam_full_tst_rmse = rmse(pts_tst$pts, predict(gam_full, pts_tst))
```

Results

Here we list all models we have used above by increasing test rmse.

	CV RMSE	TEST RMSE
GAM Full Model	0.7560024	0.8877176
Linear Full Model	0.8369245	0.8894252

	CV RMSE	TEST RMSE
Elastic Full Model	0.8043274	0.9020048
Elastic Huge Model	0.8216759	0.903584
Elastic Small Model	0.8358448	0.9041958
Elastic Interact Model	0.8282475	0.9056252
Linear Small Model	0.8374507	0.908642
Linear Interact Model	0.814104	0.9110333
GAM Small Model	0.8357086	0.9112571
Random Forest Interact Model	0.9708104	1.0122264
Random Forest Small Model	0.9008632	1.04937

As we can see, with the lowest test rmse among all, GAM full model performs the best among all the models we consider.

```
gam_full
```

```
## Generalized Additive Model using Splines
##
## 5108 samples
##   25 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 4087, 4086, 4087, 4087, 4085
## Resampling results across tuning parameters:
##
##   df  RMSE       Rsquared
##    1  0.7568492  0.9936669
##    2  0.7560024  0.9936881
##    3  0.7571414  0.9936890
##    4  0.7595504  0.9936746
##    5  0.7621311  0.9936547
##    6  0.7644795  0.9936343
##    7  0.7666425  0.9936138
##    8  0.7686974  0.9935930
##    9  0.7707073  0.9935715
##   10  0.7727396  0.9935487
##
## RMSE was used to select the optimal model using  the smallest value.
## The final value used for the model was df = 2.
```

```
get_best_result(gam_full)
```

```
##   df      RMSE  Rsquared    RMSESD  RsquaredSD
## 1  2 0.7560024 0.9936881 0.3862915 0.006907204
```

Discussion

Methods perspective

As we can see from `results`, although GAM full model outperforms the others, generally we can find out that linear methods including linear regression and penalized linear regression (elastic net) outperform non-linear

methods like trees. This actually makes sense if we go back to check the feature plot of all variables. From the feature plot, we can see that most of the variables have a somehow linear relationship with the response variable `pts`. So we believe and confirm that indeed, linear methods work better for this dataset. And GAM full model can outperform in this case because it allows for flexible nonlinearities in several variables, but still retains the additive structure of linear models, which matches what we observe from the above plots.

Features perspective

As we can see from `results`, generally we can see that full model outperforms other models with subsets of features. That is, when we consider all features in the dataset, instead of subsetting only the significant features we observe from the plots, the models will perform better with respect to predictions.

This actually makes sense if we go back to check the feature plot of all variables. From the feature plot, actually almost all variables somehow contribute to the response variable `pts`, that is, the red lines in the plot are not flat. So we should expect generally full model should outperform other models with subsets of features.