Auto Miles-per-Gallon Predictive Modeling

Phase II: Data Modelling

Akshay Sharma [redacted] *& Vishesh Jain* [redacted]

11 June 2018

Contents

1 INTRODUCTION

The objective of this project is to build a model to predict miles-per-gallon of a car. The data set was sourced from the UCI Machine Learning Repository. This project has two phases. In Phase I, we performed data cleaning, transformation & summarization and produced a data set for modeling. In Phase II, we have built three regression (error-based) models on the processed data.

This Report later contains Feature Selection, Hyper Parameter Tuning, Data Modeling and Evaluation.

2 METHODOLOGY

We considered three regression models - Kernel SVM, Lasso and K-Nearest Neighbour Regression (KKNN) and earth (multivariate adaptive regression splines). Each model was trained to make transformed miles-per-gallon prediction to check the performance based on Mean Square Error, Mean absolute Error and R-square values. The data set was split into 75% as training set and 25% as test set. For fine tuning process, we ran a three-fold cross validation on each model.

Next, for each model we performed feature selection and tuned the hyper-parameters. Using the features selected and hyper-parameter values we made predictions on the test data. During model training (feature selection and Hyper-Parameter tuning), we relied on Mean Squared Error. In addition to MSE, we also used MAE and R-Sqauare values to evaluate models' performance. The feature selection was done using `spFSR` package and modeling was done using 'mlr' package.

3 R-LIBRARY

Following packages are used in the given analysis:

```
library(caret)
```

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

```
## Loading required package: ggplot2
```

```
library(mlr)
```

```
## Loading required package: ParamHelpers
```

```
##
## Attaching package: 'mlr'
```

```
## The following object is masked from 'package:caret':
##
##     train
```

```
library(mlbench)
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------- tidyverse 1.2
```

```
## v tibble  1.4.2     v purrr   0.2.4
## v tidyr   0.8.1     v dplyr   0.7.4
## v readr   1.1.1     v stringr 1.2.0
## v tibble  1.4.2     v forcats 0.3.0
```

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

```
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
```

```
library(spFSR)
```

```
## Loading required package: parallelMap
```

```
## Loading required package: parallel
```

```
## Loading required package: tictoc
```

```
library(dplyr)
```

4 DATA SPLIT

Prior to any activity of data modeling, we will first split the data into test and training sets.

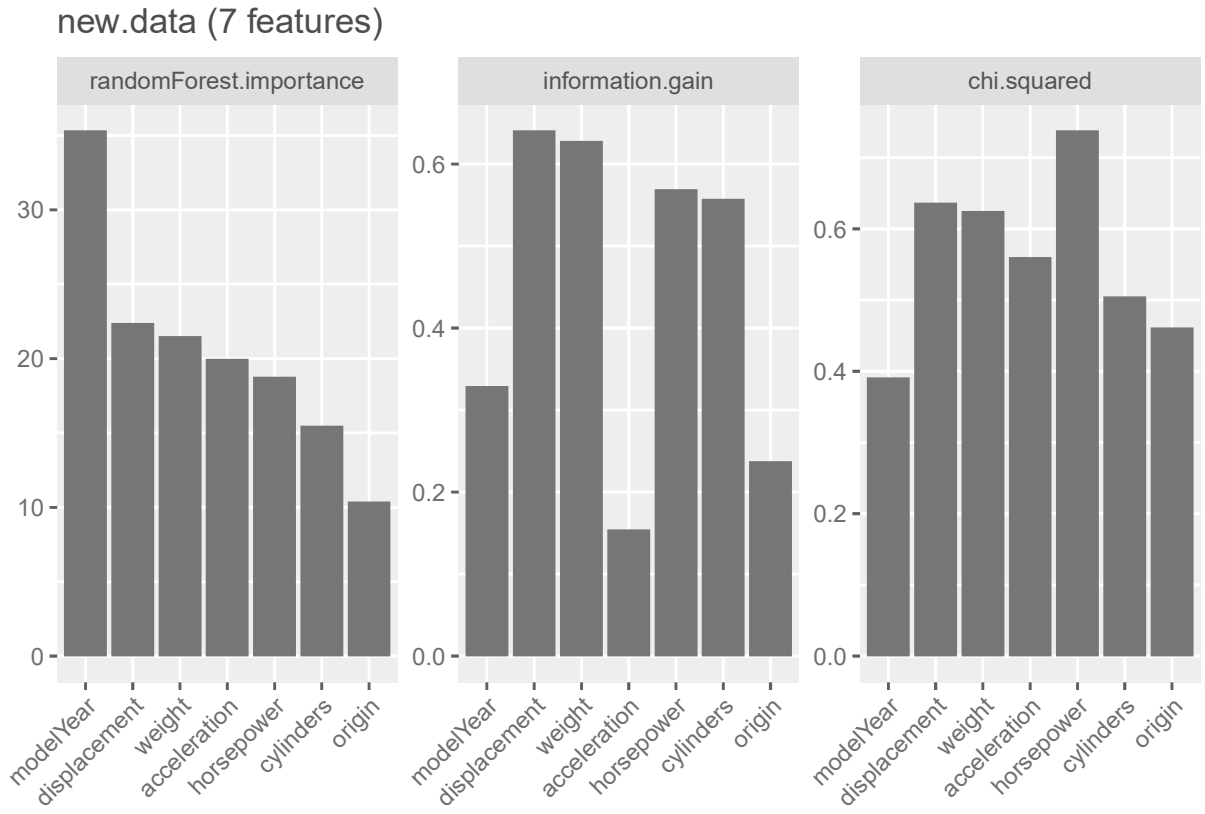
```
ampg.2 <- read.csv("ampg_transform.csv")
data <- ampg.2[,-c(1,9,10)]
data$cylinders<- as.factor(data$cylinders)
data$origin<- as.factor(data$origin)
data$modelYear<- as.factor(data$modelYear)

intrain<-createDataPartition(y=data$tmpg,p=0.75,list=FALSE)
train<-data[intrain,]
test<-data[-intrain,]


Y.train <- train %>% pull(tmpg)
X.train <- train[,-8]

Y.test <- test %>% pull(tmpg)
X.test <- test[,-8]

new.data = cbind(X.train, Y.train)
my.task <- makeRegrTask(data = new.data, target = "Y.train")
```

5 FEATURE SELECTION

In this section, we used `spFSR` package to produce feature ranking and perform feature selection using each model.

Just a look at model performance through various importance criteria using all features.

```
mFV <- generateFilterValuesData(my.task,
                                method = c('randomForest.importance',
                                           'information.gain',
                                           'chi.squared'))
plotFilterValues(mFV)
```




```
my.measure <- mse

my.wrapper1 <- makeLearner("regr.ksvm", id = 'ksvm')
my.wrapper2 <- makeLearner("regr.earth", id = 'erth')
my.wrapper3 <- makeLearner("regr.kknn", id = 'kknn')
```

```
## Loading required package: kknn
```

```
## Warning: package 'kknn' was built under R version 3.4.4
```

```
##
## Attaching package: 'kknn'
```

```
## The following object is masked from 'package:caret':
##
##     contr.dummy
```

5.1 Kernel - SVM

```
# Feature selection for ksvm
spsaMod.ksvm <- spFeatureSelection(task = my.task,
                                   wrapper = my.wrapper1,
                                   num.features.selected = 0,
                                   measure = my.measure)
```

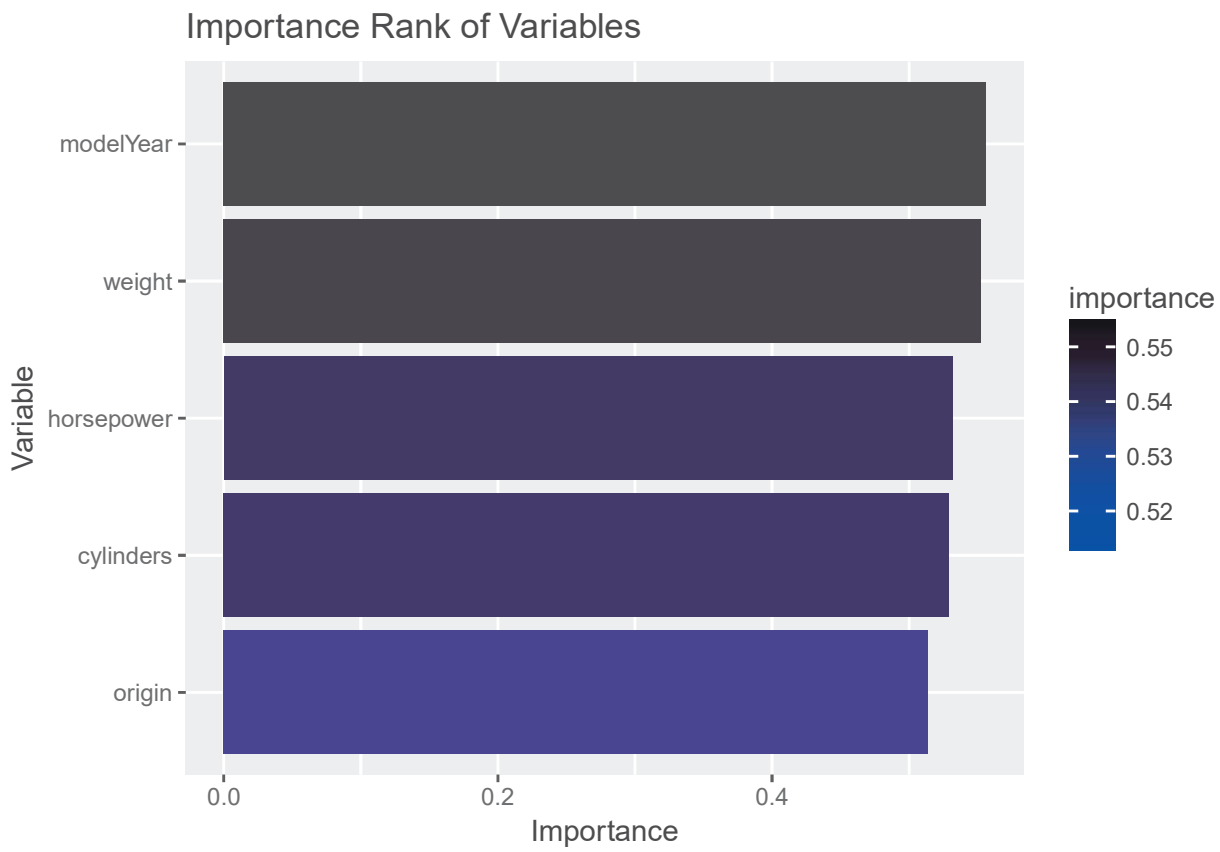
```
## Warning in spsaKernel(task = task, wrapper = wrapper, measure = measure, :
## Stratification for learner of type regr is not supported.
```

```
## Warning in spsaKernel(task = task, wrapper = wrapper, measure = measure, :
## cv.stratify is reset to FALSE.

## SPSA-FSR begins:

## Wrapper = ksvm

## Measure = mse

## Number of selected features = 0

##
## Best iteration = 13

## Number of selected features = 5

## Best measure value = 0.04979

## Std. dev. of best measure = 0.01328

## Run time = 0.74 minutes.
```

```
getImportance(spsaMod.ksvm)
```

```
##     features importance
## 1  modelYear    0.55577
## 2     weight    0.55209
## 3 horsepower    0.53153
## 4  cylinders    0.52879
## 5     origin    0.51375
```

```
plotImportance(spsaMod.ksvm)
```



```r
X.ksvm <- X.train[ , which(names(X.train) %in% as.vector(spsaMod.ksvm$features))]

print("Selected features are: ")
```

```
## [1] "Selected features are: "
```

```r
print(spsaMod.ksvm$features)
```

```
## [1] "modelYear"  "weight"     "horsepower" "cylinders"  "origin"
```

5.2 Earth

```r
spsaMod.erth <- spFeatureSelection(task = my.task,
                                   wrapper = my.wrapper2,
                                   num.features.selected = 0,
                                   measure = my.measure)
```

```
## Warning in spsaKernel(task = task, wrapper = wrapper, measure = measure, :
## Stratification for learner of type regr is not supported.
```

```
## Warning in spsaKernel(task = task, wrapper = wrapper, measure = measure, :
## cv.stratify is reset to FALSE.
```

```
## SPSA-FSR begins:
```

```
## Wrapper = earth
```

```
## Measure = mse
```

```
## Number of selected features = 0
```

```
##
## Best iteration = 19
```

```
## Number of selected features = 3
```

```
## Best measure value = 0.04614
```

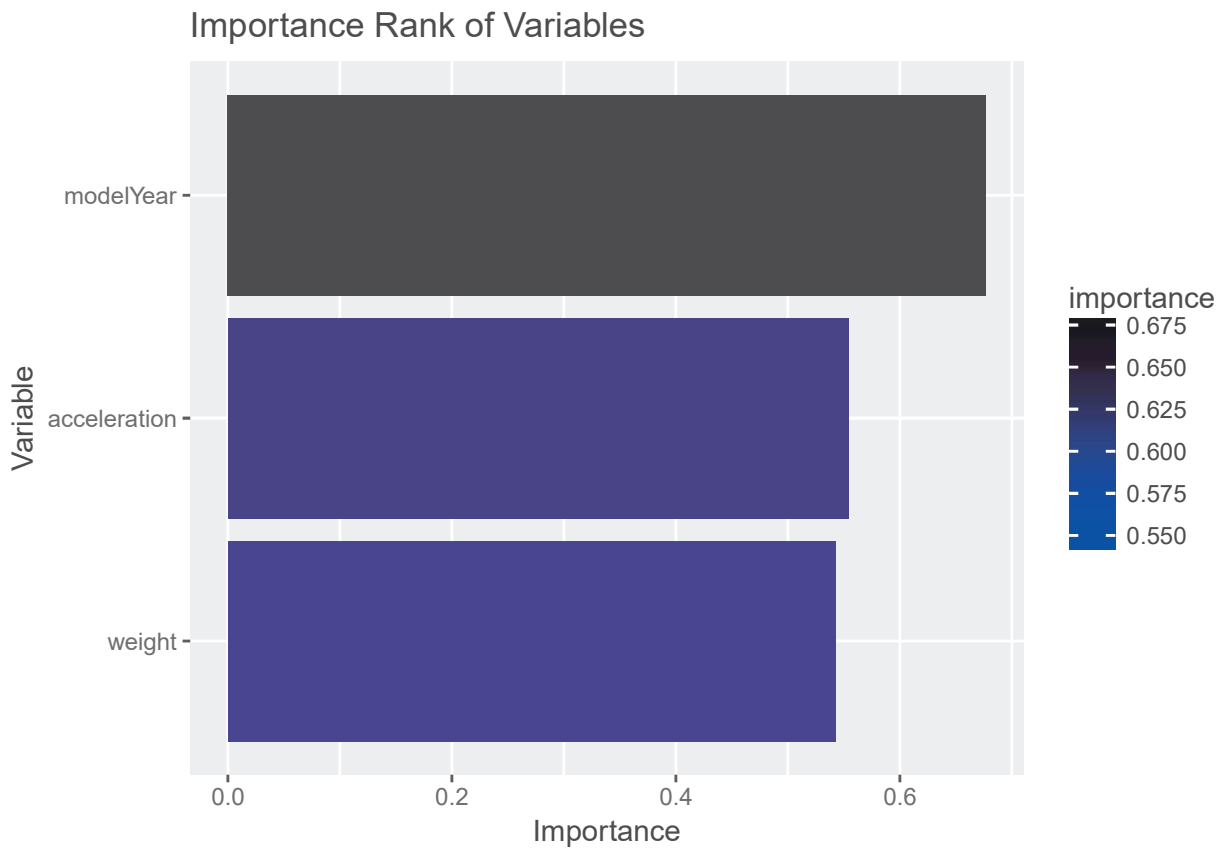
```
## Std. dev. of best measure = 0.01139
```

```
## Run time = 0.8 minutes.
```

```r
getImportance(spsaMod.erth)
```

```
##       features importance
## 1    modelYear    0.67664
## 2 acceleration    0.55388
## 3       weight    0.54221
```

```r
plotImportance(spsaMod.erth)
```




```
X.erth <- X.train[ , which(names(X.train) %in% as.vector(spsaMod.erth$features))]


print("Selected features are: ")
```

```
## [1] "Selected features are: "
```

```
print(spsaMod.erth$features)
```

```
## [1] "modelYear"    "acceleration" "weight"
```

5.3 KKNN

```
spsaMod.kknn <- spFeatureSelection(task = my.task,
                                   wrapper = my.wrapper3,
                                   num.features.selected = 0,
                                   measure = my.measure)
```

```
## Warning in spsaKernel(task = task, wrapper = wrapper, measure = measure, :
## Stratification for learner of type regr is not supported.
```

```
## Warning in spsaKernel(task = task, wrapper = wrapper, measure = measure, :
## cv.stratify is reset to FALSE.
```

```
## SPSA-FSR begins:
```

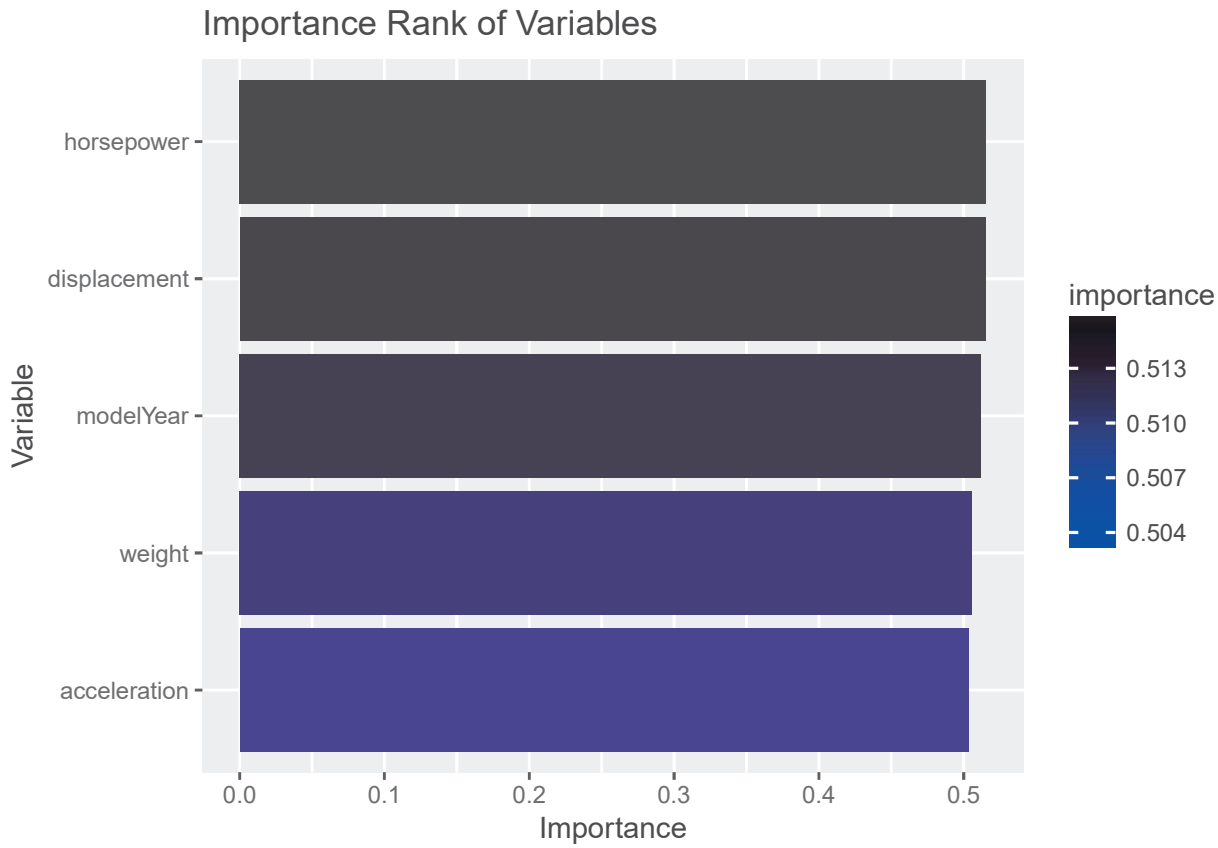
```
## Wrapper = kknn
```

```
## Measure = mse

## Number of selected features = 0

##
## Best iteration = 4

## Number of selected features = 5

## Best measure value = 0.05124

## Std. dev. of best measure = 0.01269

## Run time = 0.35 minutes.
```

```
getImportance(spsaMod.kknn)
```

```
##       features importance
## 1   horsepower    0.51560
## 2 displacement    0.51494
## 3    modelYear    0.51197
## 4       weight    0.50561
## 5 acceleration    0.50311
```

```
plotImportance(spsaMod.kknn)
```




```
X.kknn <- X.train[ , which(names(X.train) %in% as.vector(spsaMod.kknn$features))]

print("Selected features are: ")
```

```
## [1] "Selected features are: "
```

```
print(spsaMod.kknn$features)
```

```
## [1] "horsepower"   "displacement" "modelYear"    "weight"
## [5] "acceleration"
```

6 HYPER-PARRAMETER TUNING

Setting the re-sampling description:

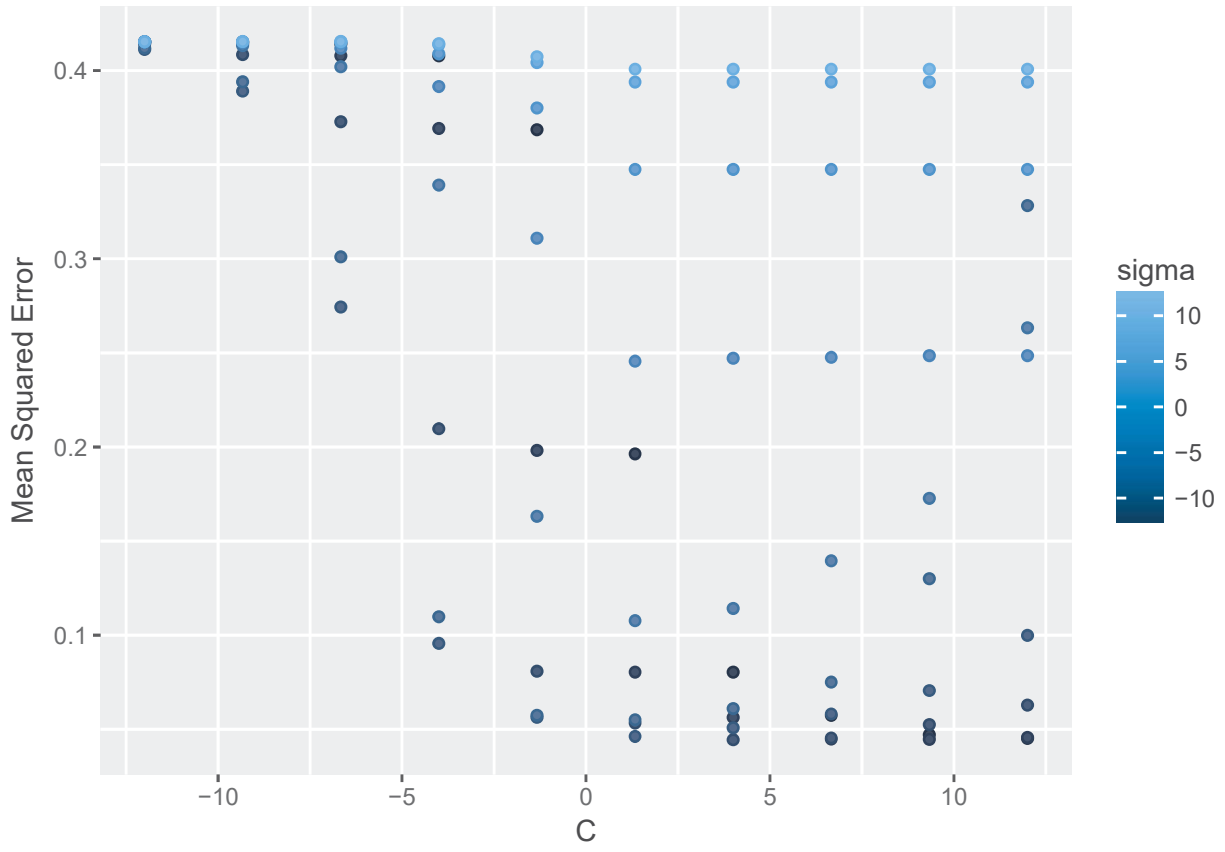
```
my.rdesc <- makeResampleDesc("RepCV", folds = 3, reps = 3, stratify = F)
```

6.1 Kernel SVM

- We are going to tune the sigma and C parameter of kernel SVM and use grid search to search for the best values for these parameters.

```
data.ksvm = cbind(X.ksvm,Y.train)
my.task.ksvm <- makeRegrTask(id = "ksvm", data = data.ksvm, target = "Y.train")
ps.ksvm = makeParamSet(makeNumericParam("C", lower = -12, upper = 12, trafo = function(x) 2^x),
                       makeNumericParam("sigma", lower = -12, upper = 12, trafo = function(x) 2^x))
ctrl.ksvm = makeTuneControlGrid()
set.seed(123, kind = "L'Ecuyer-CMRG")
res.ksvm = tuneParams("regr.ksvm", task = my.task.ksvm, resampling = my.rdesc,
                      par.set = ps.ksvm, control = ctrl.ksvm, show.info = F)
```

```
data.ksvm = generateHyperParsEffectData(res.ksvm)
plt.ksvm = plotHyperParsEffect(data.ksvm, x = "C", y = "mse.test.mean", z = 'sigma')
plt.ksvm + ylab("Mean Squared Error")
```



Based on above results, we can see that the tuned parameters are :

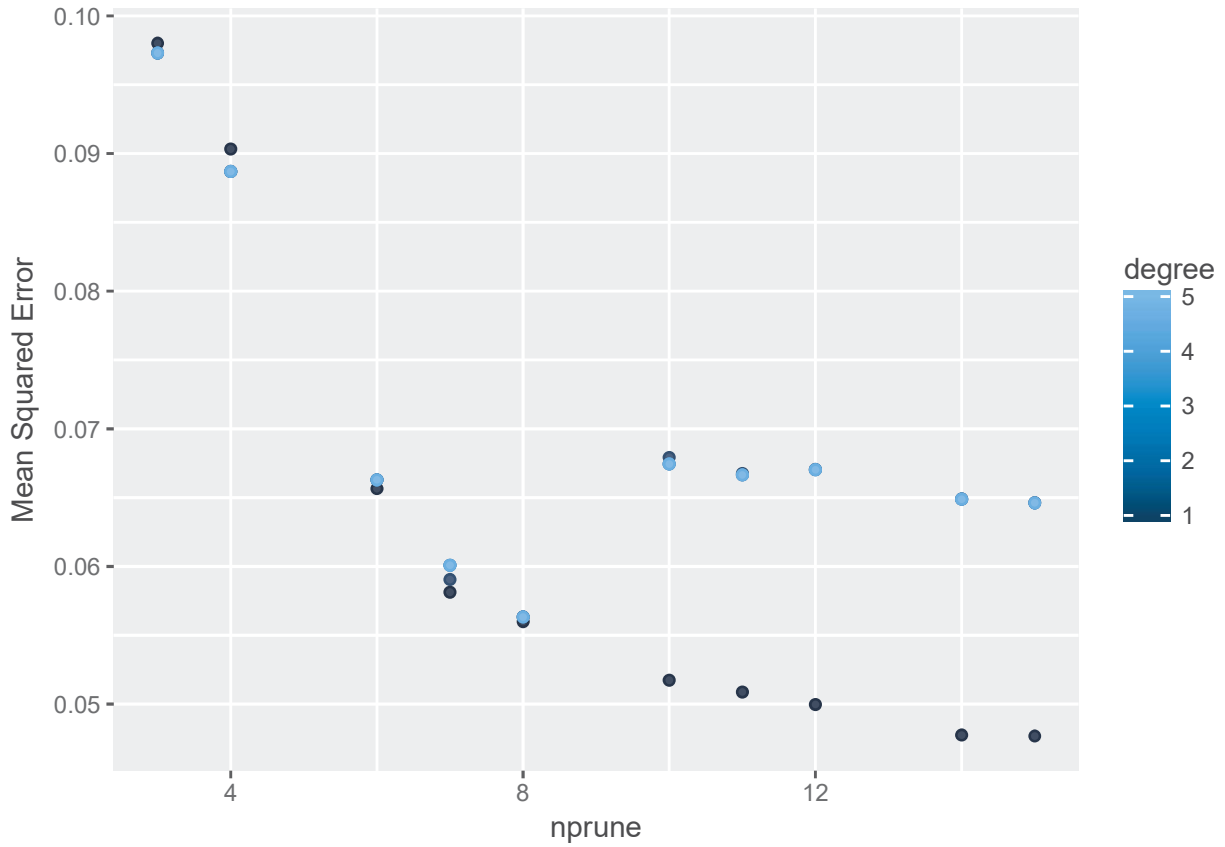
C=4.1e+03; sigma=0.000244 : mse.test.mean=0.0443031

6.2 EARTH

- We are going to tune the degree of interactions and nprune i.e. number of parameters to be pruned of Earth algorithm and use grid search to search for the best values for these parameters.

```
data.erth = cbind(X.erth,Y.train)
my.task.erth <- makeRegrTask(id = "earth", data = data.erth, target = "Y.train")
ps.erth = makeParamSet(makeIntegerParam("nprune", lower = 3, upper = 15),
                       makeIntegerParam("degree", lower = 1, upper = 5))
ctrl.erth = makeTuneControlGrid()
set.seed(123, kind = "L'Ecuyer-CMRG")
res.erth = tuneParams("regr.earth", task = my.task.erth, resampling = my.rdesc,
                      par.set = ps.erth, control = ctrl.erth, show.info = F)
```

```
data.erth = generateHyperParsEffectData(res.erth)
plt.erth = plotHyperParsEffect(data.erth, x = "nprune", y = "mse.test.mean", z = 'degree')
plt.erth + ylab("Mean Squared Error")
```



Based on above graph, we obtained the following result for the parameter:

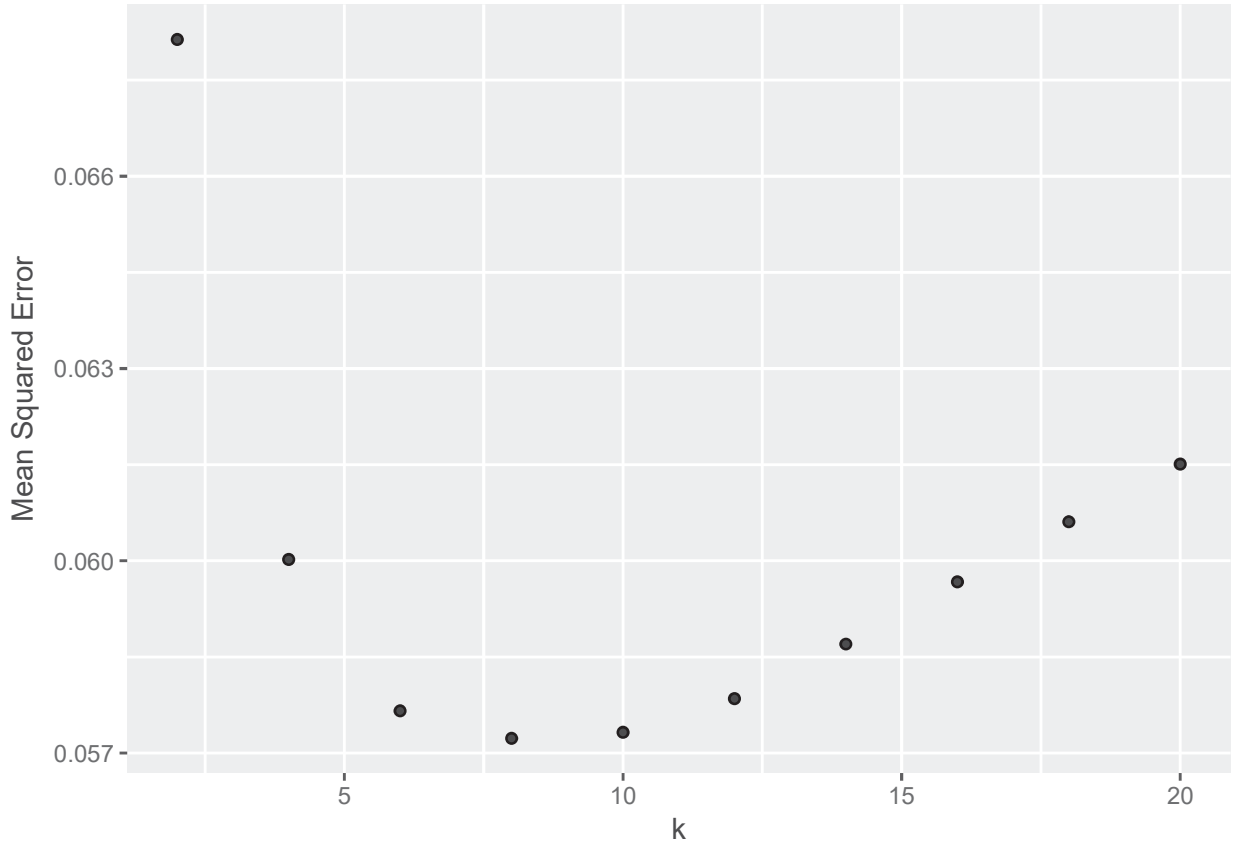
nprune=15; degree=1 : mse.test.mean=0.0482361

6.3 KKNN

- We are going to tune the k parameter i.e. number of neighbours to be taken into account for k nearest neighbour regression technique and used grid search to search for the best values of k parameter.

```
data.kknn = cbind(X.kknn,Y.train)
my.task.kknn <- makeRegrTask(id = "kknn", data = data.kknn, target = "Y.train")
ps.kknn = makeParamSet(makeNumericParam("k",lower = 2,upper = 20))
ctrl.kknn = makeTuneControlGrid()
set.seed(123, kind = "L'Ecuyer-CMRG")
res.kknn = tuneParams("regr.kknn", task = my.task.kknn, resampling = my.rdesc,
                      par.set = ps.kknn, control = ctrl.kknn, show.info = F)
```

```
data.kknn = generateHyperParsEffectData(res.kknn)
plt = plotHyperParsEffect(data.kknn, x = "k", y = "mse.test.mean")
plt + ylab("Mean Squared Error")
```



Based on above graphs, we found the following values of tuned parameters:

k = 6 : mse.test.mean=0.052

7 MODEL TRAINING

In thi section we will train the model based on the features selected and tuned hyper-parameters.

7.1 Kernel SVM

```
lrn.ksvm = setHyperPars(makeLearner("regr.ksvm"), par.vals = res.ksvm$x)
model.ksvm = mlr::train(lrn.ksvm, my.task.ksvm)

# Cross-Validation
set.seed(123, kind = "L'Ecuyer-CMRG")
repcv.ksvm <- resample(lrn.ksvm,
                       my.task.ksvm,
                       my.rdesc,
                       measures = my.measure)
```

```
## [Resample] repeated cross-validation iter 1: mse.test.mean=0.051
## [Resample] repeated cross-validation iter 2: mse.test.mean=0.0412
## [Resample] repeated cross-validation iter 3: mse.test.mean=0.047
## [Resample] repeated cross-validation iter 4: mse.test.mean=0.0504
```

```
## [Resample] repeated cross-validation iter 5: mse.test.mean=0.0349
## [Resample] repeated cross-validation iter 6: mse.test.mean=0.0465
## [Resample] repeated cross-validation iter 7: mse.test.mean=0.0405
## [Resample] repeated cross-validation iter 8: mse.test.mean=0.0466
## [Resample] repeated cross-validation iter 9: mse.test.mean=0.0418
## [Resample] Aggr. Result: mse.test.mean=0.0444
```

7.2 Earth

```
lrn.erth = setHyperPars(makeLearner("regr.earth"), par.vals = res.erth$x)
model.earth = mlr::train(lrn.erth, my.task.erth)

# Cross-Validation
repcv.erth <- resample(lrn.erth,
                       my.task.erth,
                       my.rdesc,
                       measures = my.measure)
```

```
## [Resample] repeated cross-validation iter 1: mse.test.mean=0.0477
## [Resample] repeated cross-validation iter 2: mse.test.mean=0.046
## [Resample] repeated cross-validation iter 3: mse.test.mean=0.0532
## [Resample] repeated cross-validation iter 4: mse.test.mean=0.0584
## [Resample] repeated cross-validation iter 5: mse.test.mean=0.0557
## [Resample] repeated cross-validation iter 6: mse.test.mean=0.038
## [Resample] repeated cross-validation iter 7: mse.test.mean=0.0493
## [Resample] repeated cross-validation iter 8: mse.test.mean=0.0644
## [Resample] repeated cross-validation iter 9: mse.test.mean=0.0379
## [Resample] Aggr. Result: mse.test.mean=0.0501
```

7.3 KKNN

```
lrn.kknn = setHyperPars(makeLearner("regr.kknn"), par.vals = res.kknn$x)
model.kknn = mlr::train(lrn.kknn, my.task.kknn)

# Cross-Validation
repcv.kknn <- resample(lrn.kknn,
                       my.task.kknn,
                       my.rdesc,
                       measures = my.measure)
```

```
## [Resample] repeated cross-validation iter 1: mse.test.mean=0.0489
## [Resample] repeated cross-validation iter 2: mse.test.mean=0.056
## [Resample] repeated cross-validation iter 3: mse.test.mean=0.0507
## [Resample] repeated cross-validation iter 4: mse.test.mean=0.0589
## [Resample] repeated cross-validation iter 5: mse.test.mean=0.0753
## [Resample] repeated cross-validation iter 6: mse.test.mean=0.0563
## [Resample] repeated cross-validation iter 7: mse.test.mean=0.0492
## [Resample] repeated cross-validation iter 8: mse.test.mean=0.055
## [Resample] repeated cross-validation iter 9: mse.test.mean=0.0658
## [Resample] Aggr. Result: mse.test.mean=0.0574
```

7.4 Model Comparison - CV

Comparing the mean square error of all models before evaluation.

```
result.ksvm.mean <- mean(repcv.ksvm$measures.test[[2]])
result.erth.mean <- mean(repcv.erth$measures.test[[2]])
result.kknn.mean <- mean(repcv.kknn$measures.test[[2]])

Model.Used <- c('ksvm','earth','kknn')
Model.Results <-c(result.ksvm.mean, result.erth.mean, result.kknn.mean)
print(data.frame(Model.Used,Model.Results))
```

```
##   Model.Used Model.Results
## 1       ksvm    0.04444004
## 2      earth    0.05005512
## 3       kknn    0.05736129
```

Prior to evaluation we can see that all algorithms are performing well however, we can clearly see that KKNN is the winner with the lowest Mean Squared Error value.

We will proceed with evaluation of all three models.

8 EVALUATION

In this section, we will use the trained models to predict target levels on the test data. Then we will use the observed and predicted values to calculate evaluation parameters.

```
data.pred = cbind(X.test,Y.test)
colnames(data.pred)[8]<-"Y.train"

pred = predict(model.ksvm, newdata = data.pred)
print("-----------------------KSVM Evaluation-------------------")
```

```
## [1] "-----------------------KSVM Evaluation-------------------"
```

```
performance(pred, measures = list(mse, mae, sse, rsq, expvar))
```

```
##        mse        mae        sse        rsq     expvar
## 0.03529037 0.14266454 3.38787550 0.90371216 0.84187685
```

```
pred = predict(model.earth, newdata = data.pred)
print("-----------------------Earth Evaluation-------------------")
```

```
## [1] "-----------------------Earth Evaluation-------------------"
```

```
performance(pred, measures = list(mse, mae, sse, rsq, expvar))
```

```
##        mse        mae        sse        rsq     expvar
## 0.04125266 0.15502602 3.96025567 0.88744437 0.86144905
```

```
pred = predict(model.kknn, newdata = data.pred)
print("-----------------------kknn Evaluation-------------------")
```

```
## [1] "-----------------------kknn Evaluation-------------------"
```

```
performance(pred, measures = list(mse, mae, sse, rsq, expvar))
```

```
##        mse        mae        sse        rsq     expvar
```

```
## 0.05096515 0.16327534 4.89265424 0.86094439 0.87879154
```

- Lowest Mean Sqaured Error, Mean Absolute Error, Sum of Sqaure Error is observed in KSVM
- Maximum R-Square Value is observed in Earth
- Maximum explained variance is given by Earth algorithm

9 DISCUSSION

The previous section showed evaluation of each regression model. We can see that all the models are performing well, as there is only ~1% error in all models. ALthough we observed that the explained variance is higher in Earth model, but, KSVM outperformed the other two models when mse, mae, sse and rsq values were taken into account.

10 CONCLUSION

Among the three regression models, KSVM produced best performance in predicting miles-per-gallon of a car. Using random sampling, we split the data into 75% train and 25% test data. We also performed feature selection and hyper-parameter tuning for each regression model. However, the explained variance is high in earth model, taking over-fitting into consideration, we would recommend KSVM.