Boston Data ml4econ Kaggle Competition

june 23 2019

by *Dor Meir and Inbal Dekel*

Loading the data

```
install.packages("readr", repos = "http://cran.us.r-project.org")
library(readr)

train <- read_csv("data/train.csv")
attach(train)
test <- read_csv("data/test.csv")
submissionExample <- read_csv("data/submissionExample.csv")
```

Acquiring domain knowledge

To acquire domain knowledge, we shall look at the documentation, structure and summary of the “Boston Housing Data”, which the train and test datasets are based on.

```
#install.packages("MASS", repos = "http://cran.us.r-project.org")
library(MASS)
#Boston?
str(Boston)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
summary(Boston)
```

```
##       crim                zn             indus            chas        
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   Min.   :0.00000  
##  1st Qu.: 0.08204   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000  
##  Median : 0.25651   Median :  0.00   Median : 9.69   Median :0.00000  
##  Mean   : 3.61352   Mean   : 11.36   Mean   :11.14   Mean   :0.06917  
##  3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000  
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.00000  
##       nox               rm             age              dis        
##  Min.   :0.3850   Min.   :3.561   Min.   :  2.90   Min.   : 1.130  
##  1st Qu.:0.4490   1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.100  
##  Median :0.5380   Median :6.208   Median : 77.50   Median : 3.207  
##  Mean   :0.5547   Mean   :6.285   Mean   : 68.57   Mean   : 3.795  
##  3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.188  
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.127  
##       rad              tax           ptratio          black       
##  Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   :  0.32  
##  1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38  
##  Median : 5.000   Median :330.0   Median :19.05   Median :391.44  
##  Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :356.67  
##  3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23  
##  Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90  
##      lstat            medv      
##  Min.   : 1.73   Min.   : 5.00  
##  1st Qu.: 6.95   1st Qu.:17.02  
##  Median :11.36   Median :21.20  
##  Mean   :12.65   Mean   :22.53  
##  3rd Qu.:16.95   3rd Qu.:25.00  
##  Max.   :37.97   Max.   :50.00
```

As can be seen, this dataset contains 506 observations and 14 variables. The response variable "medv" represents Boston area median house values and the predictors are a set of area specific features. The only factor variable is "chas" and it's already coded as 0/1. There doesn't seem to be a justification for creating interaction terms with this variable.

Exploring and pre-processing the data

First, let us check if there are any missing values in the train and test datasets.

```
which(is.na(train))
```

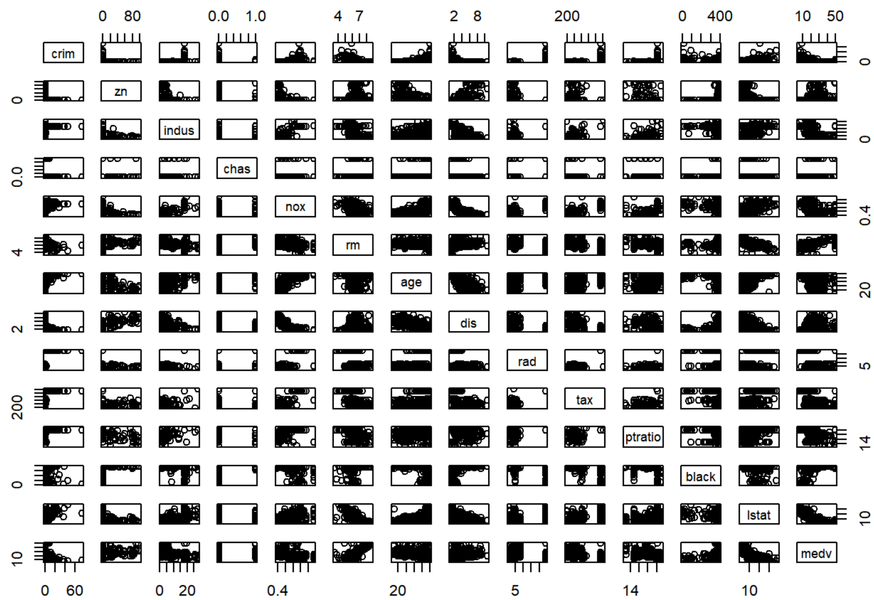
```
## integer(0)
```

```
which(is.na(test))
```

```
## integer(0)
```

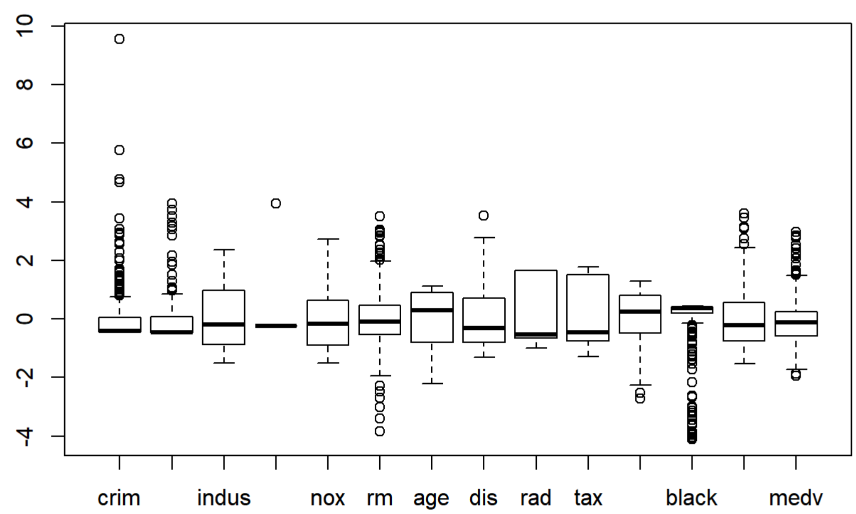
As can be seen, there are no missing values. Now, let us create a scatter plot of every two variables (except "ID") in the train dataset.

```
plot(train[-1])
```



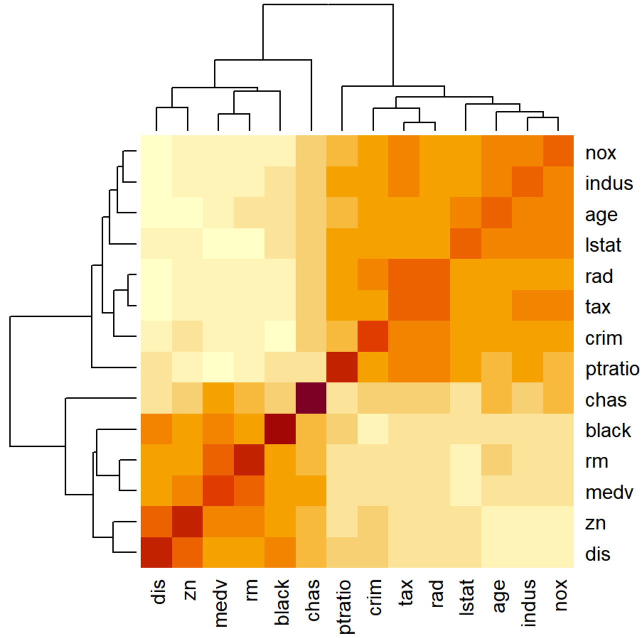
As can be seen, some of the variables seems to have complex and non-linear relationships. Now let us scale the variables in the train dataset (that is, subtract their mean and divide by their standard error), and create a box plot of the scaled variables.

```
train_scaled <- scale(train[-1])
boxplot(train_scaled)
```



As can be seen, the medians of most scaled variables in the train dataset are close to zero. Moreover, the variables “crim”, “zn”, “rm” and “black” seem to have many outliers. Now let us create a heat-map of the correlations between any two scaled variables (except “ID”) in the train dataset.

```
heatmap(cor(train_scaled))
```



As can be seen, the features with the largest variances are “dis”, “zn”, “rm”, “black”, “chas”, “ptratio” and “crime”. Yet whereas the response variable “medv” is highly correlated with the first five features, it is not correlated with “ptratio” and “crime”. This implies that PLS may be better than PCR in our context. This is because the underlying assumption in PCR is that the directions in which the features vary the most are also linked to the dependent variable, and this doesn’t seem to be the case here. That is, PCR might overweight the variables “ptratio” and “crime” in the construction of the PCs.

Choosing a model class:

Let us go over the ML methods that we have studied in class to see which of them seems most appropriate for use in our setting.

- Since the response variable (medv) is continuous, classification methods are irrelevant.
- KNN is typically good for either one or two variables. Otherwise, the number of observations needs to grow exponentially. Since our train dataset consists of 13 features and only 333 observations, KNN doesn’t seems to be appropriate in our setting.
- Unsupervised learning doesn’t seems to be appropriate in our setting since we do have a response variable (medv).

Hence, it seems justifiable to use dimension reduction and tree-based methods. Specifically,

Dimention reduction methods:

- We will use Elastic Nets and find the optimal alpha (together with the optimal lambda) using Cross Validation.
- We will prefer PLS over PCR since it seems from the heat-map that the scaled variables “ptratio” and “crime” have high variances but that they are not correlated with the scaled response variable “medv”. Thus PCR, which computes PCs in an unsupervised manner, might overweight these two variables.

Tree-based methods:

Since the scatter plot implies that there are complex and non-linear relationships between variables, it seems appropriate to use tree-based methods. Specifically, out of these methods, Random Forests seem best:

- Random Forests seem better than a single decision tree since they combine a large number of trees and may thus reduce the variance and improve prediction accuracy. While combining a large number of trees causes a loss in interpretation, it doesn’t matter to us as all we need in this task is to predict the response variable.
- Random Forests seem better than Bagging since they decorrelate the trees, and may thus reduce the variance of the average of trees while keeping its bias the same.
- Random Forests may be better than Boosting since choosing a wrong number of iterations in Boosting might lead to over-fitting.

Hence our chosen methods, that seem to be most appropriate in our setting, are: Elastic Nets, PLS and Random Forests. After we estimate these methods, we will create an ensemble (i.e., a weighted average) of the resulting predictions, where the weights will be estimated using Boosting.

Estimating the selected model classes and deriving predictions

Elastic Net:

First, let us install and use glmnbetUtils to produce Elastic Net Cross-Validation for alpha and lambda simultaneously.

```
install.packages("glmnetUtils", repos = "http://cran.us.r-project.org")
library(glmnetUtils)
elastic_net <- cva.glmnet(medv ~ . - ID, data = train)
```

Note that the following defaults have been used: (1) A sequence of 11 values more closely spaced around 0 were used as alpha values for which to do Cross-Validation; (2) The number of Cross-Validation folds was 10; (3) All predictors were standardized prior to fitting the model.

Now we shall find the best alpha and lambda that minimize the Cross-Validation MSE.

```
install.packages("data.table", repos = "http://cran.us.r-project.org")
library(data.table)

num_alphas <- length(elastic_net$alpha)
table <- data.table()

for (i in 1:num_alphas){
  alpha <- elastic_net$alpha[i] # A given alpha
  min_lambda <- elastic_net$modlist[[i]]$lambda.min # Lambda that minimizes CV-MSE for
 the given alpha
  min_mse <-  min(elastic_net$modlist[[i]]$cvm) # The minimum value of CV-MSE over lamb
das for the given alpha

  new_row <- data.table(alpha, min_lambda, min_mse)
  table <- rbind(table, new_row)
}

best_alpha_lambda <- table[which.min(table$min_mse)]
colnames(best_alpha_lambda) <- c("Optimal alpha", "Optimal lambda", "CV-MSE")
best_alpha  <- c(as.matrix(best_alpha_lambda[1,"Optimal alpha"]))
best_lambda <- c(as.matrix(best_alpha_lambda[1,"Optimal lambda"]))
```

And the optimal alpha and lambda (together with the minimized CV-MSE) are:

```
best_alpha_lambda
```

```
##    Optimal alpha Optimal lambda  CV-MSE
## 1:         0.008     0.08456678 25.6122
```

Using these optimal parameters, we can predict the response for the test and train datasets.

```
predicted_test_elastic_net  <- predict(elastic_net, s = best_lambda, alpha = best_alph
a, newdata = test[-1])
predicted_train_elastic_net <- predict(elastic_net, s = best_lambda, alpha = best_alph
a, newdata = train[-c(1,15)])
```

PLS:

First we shall find the number of PCs that minimizes the Cross-Validation MSE.

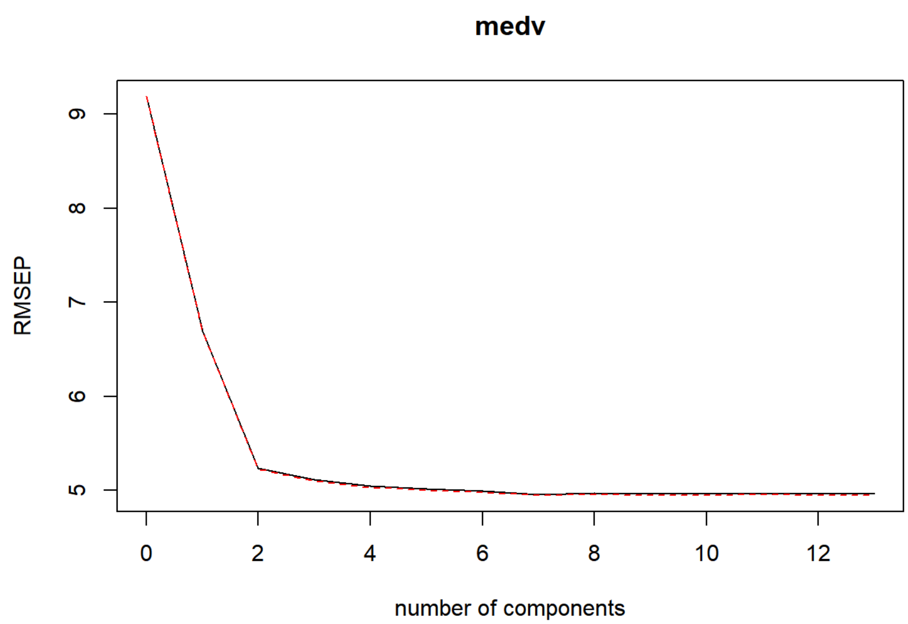
```
install.packages("pls", repos = "http://cran.us.r-project.org")
library(pls)
```

```
pls <- plsr(medv ~ . - ID, data = train, scale = TRUE, validation = "CV")

summary(pls)
```

```
## Data:    X dimension: 333 13
##  Y dimension: 333 1
## Fit method: kernelpls
## Number of components considered: 13
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           9.187    6.694    5.237    5.115    5.046    5.019    4.992
## adjCV        9.187    6.691    5.230    5.104    5.033    5.005    4.978
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       4.961    4.974    4.964     4.966     4.967     4.967     4.967
## adjCV    4.949    4.961    4.952     4.953     4.954     4.954     4.954
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X       46.40    56.86    64.03    69.81    75.71    78.55    81.54
## medv    47.74    69.17    71.44    72.49    72.91    73.15    73.26
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## X       85.11    89.55     93.09     96.13     98.60    100.00
## medv    73.30    73.31     73.31     73.31     73.31     73.31
```

```
validationplot(pls)
```



Looking at the summary and plot, it is evident that the number of components that minimizes the RMSEP is 10. Yet if we look at the percentage of explained variance that each component adds (or the decrease in RMSEP due to the addition of any component), it seems that 5 components are enough. Hence we will use 5 components to predict the response for the test and train datasets.

```
predicted_train_pls <- predict(pls, newdata = train, ncomp = 5)
predicted_test_pls  <- predict(pls, newdata = test, ncomp = 5)
```

Random Forest:

```
install.packages("randomForest", repos = "http://cran.us.r-project.org")
library(randomForest)
```

```
random_forest <- randomForest(medv ~ . - ID, data = train)

random_forest
```

```
##
## Call:
##  randomForest(formula = medv ~ . - ID, data = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 11.02171
##                     % Var explained: 86.86
```

We can now predict the response for the test and train datasets given our estimates.

```
predicted_test_random_forest  <- predict(random_forest, newdata = test)
predicted_train_random_forest <- predict(random_forest, newdata = train)
```

Ensemble:

Now we shall create an ensemble (i.e., a weighted average) of the derived predictions. To determine the weights, we will use Boosting and derive the relative importance of the predictors.

```
install.packages("gbm", repos = "https://CRAN.R-project.org")
library(gbm)

boosting <- gbm(train$medv ~ predicted_train_elastic_net + predicted_train_pls + predic
ted_train_random_forest - 1, distribution="gaussian")
weights <- as.matrix(summary(boosting, plotit = FALSE)[2])
weights <- weights / sum(weights)
```

The resulting weights are:

```
weights
```

```
##                                    rel.inf
## predicted_train_random_forest 0.9987155150
## predicted_train_elastic_net   0.0010922002
## predicted_train_pls           0.0001922848
```

As can be seen, the Random Forest predictor receives most of the weight. Now we shall use these weights to create our ensemble prediction (over the test dataset).

```
X <- cbind(predicted_test_elastic_net, predicted_test_pls, predicted_test_random_fores
t)
ensemble <- X[,1] * weights[2] + X[,2] * weights[3] + X[,3] * weights[1]

submission <- data.frame(test$ID, ensemble)
colnames(submission) <- c("ID", "medv")
write.csv(submission, file = "submission.csv",row.names=FALSE)
```