# Lecture 4: Applications A

Yufei Yin

## A. Feature Engineering

### 1.

```
data("airquality")
AQ = na.omit(airquality[,1:4])
AQ$TWcp = with(AQ, Temp * Wind)
AQ$TWrat = with(AQ, Temp / Wind)
```

```
summary(AQ$TWcp)[c(1,4,6)]
```

```
##     Min.     Mean     Max.
##  216.200  756.527 1490.400
```

The minimum value for variable `TWcp` is 216.200. The mean value for variable `TWcp` is 756.527. The maximum value for variable `TWcp` is 1490.400.

```
summary(AQ$TWrat)[c(1,4,6)]
```

```
##      Min.      Mean      Max.
##  3.034826  9.419117 40.869565
```

The minimum value for variable `TWrat` is 3.034826. The mean value for variable `TWrat` is 9.419117. The maximum value for variable `TWrat` is 40.869565.

### 2.

### (a)

```
fit.cp = lm(Ozone ~ Temp + Wind + TWcp, data = AQ)
summary(fit.cp)$coefficients[4,3]
```

```
## [1] -3.986851
```

The t-test result for variable `TWcp` is -3.986851.

```
fit.rat = lm(Ozone ~ Temp + Wind + TWrat, data = AQ)
summary(fit.rat)$coefficients[4,3]
```

```
## [1] 4.005288
```

The t-test for variable `TWrat` is 4.005288.

## (b)

```
summary(fit.cp)
```

```
##
## Call:
## lm(formula = Ozone ~ Temp + Wind + TWcp, data = AQ)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -40.930 -11.193  -3.034   8.193  97.456
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -239.8918    48.6200  -4.934 2.97e-06 ***
## Temp           4.0005     0.5935   6.741 8.26e-10 ***
## Wind          13.5975     4.2835   3.174 0.001961 **
## TWcp          -0.2173     0.0545  -3.987 0.000123 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 20.37 on 107 degrees of freedom
## Multiple R-squared:  0.6355, Adjusted R-squared:  0.6253
## F-statistic: 62.19 on 3 and 107 DF,  p-value: < 2.2e-16
```

```
summary(fit.rat)
```

```
##
## Call:
## lm(formula = Ozone ~ Temp + Wind + TWrat, data = AQ)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -55.241 -10.969  -3.506  11.568  80.805
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -85.5258    22.5920  -3.786 0.000253 ***
## Temp          1.4214     0.2557   5.559 2.01e-07 ***
## Wind         -0.6654     0.9090  -0.732 0.465756
## TWrat         2.5121     0.6272   4.005 0.000115 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 20.36 on 107 degrees of freedom
## Multiple R-squared:  0.636,  Adjusted R-squared:  0.6258
## F-statistic: 62.31 on 3 and 107 DF,  p-value: < 2.2e-16
```

We compare the corresponding p-value to significant level $\alpha = 0.05$, both of the p-values are less than $\alpha$, so we reject the null hypothesis, therefore, both `TWcp` and `TWrat` are statistically significant.

### (c)

The model is $f(X) = \beta_0 + \beta_1\, Temp + \beta_2\, Wind + \beta_3\, Temp \times Wind$. For a fixed value of variable `Wind`, the slope of `Temp` is $\beta_1 + Wind\, \beta_3$

```
coef(fit.cp)
```

```
## (Intercept)        Temp        Wind        TWcp
## -239.891827    4.000528   13.597458   -0.217276
```

```
min(AQ$Wind)
```

```
## [1] 2.3
```

```
4.000528 + 2.3 * -0.217276
```

```
## [1] 3.500793
```

When `Wind` is at its minimum value 2.3, the slope of the `Temp` effect is 3.500793.

```
max(AQ$Wind)
```

```
## [1] 20.7
```

```
4.000528 + 20.7 * -0.217276
```

```
## [1] -0.4970852
```

When `Wind` is at its maximum value 20.7, the slope of the `Temp` effect is -0.4970852.

## 3.

```
### Create function to compute MSPEs
get.MSPE = function(Y, Y.hat){
return(mean((Y - Y.hat)^2))
}

set.seed(4099183)

n = nrow(AQ) # Sample size
reorder = sample.int(n) # Randomized order

### Identify which observations go in the training set (set == 1) or the
### validation set (set == 2)
prop.train = 0.75 # Proportion of observations to put in the training set
set = ifelse(test = (reorder < prop.train * n),
             yes = 1,
             no = 2)

### Construct training and validation sets
data.train = AQ[set == 1, ]
data.valid = AQ[set == 2, ]
Y.valid = data.valid$Ozone

### Fit both models to the training set
fit.cp = lm(Ozone ~ . - TWrat, data = data.train)
fit.rat = lm(Ozone ~ . - TWcp, data = data.train)
```

```r
### Get predictions
pred.cp = predict(fit.cp, data.valid)
pred.rat = predict(fit.rat, data.valid)

### Get MSPEs
MSPEs = rep(0, times=2)
names(MSPEs) = c("CP", "Ratio")
MSPEs["CP"] = get.MSPE(Y.valid, pred.cp)
MSPEs["Ratio"] = get.MSPE(Y.valid, pred.rat)
print(signif(MSPEs, 3))
```

```
##    CP Ratio
##   199   218
```

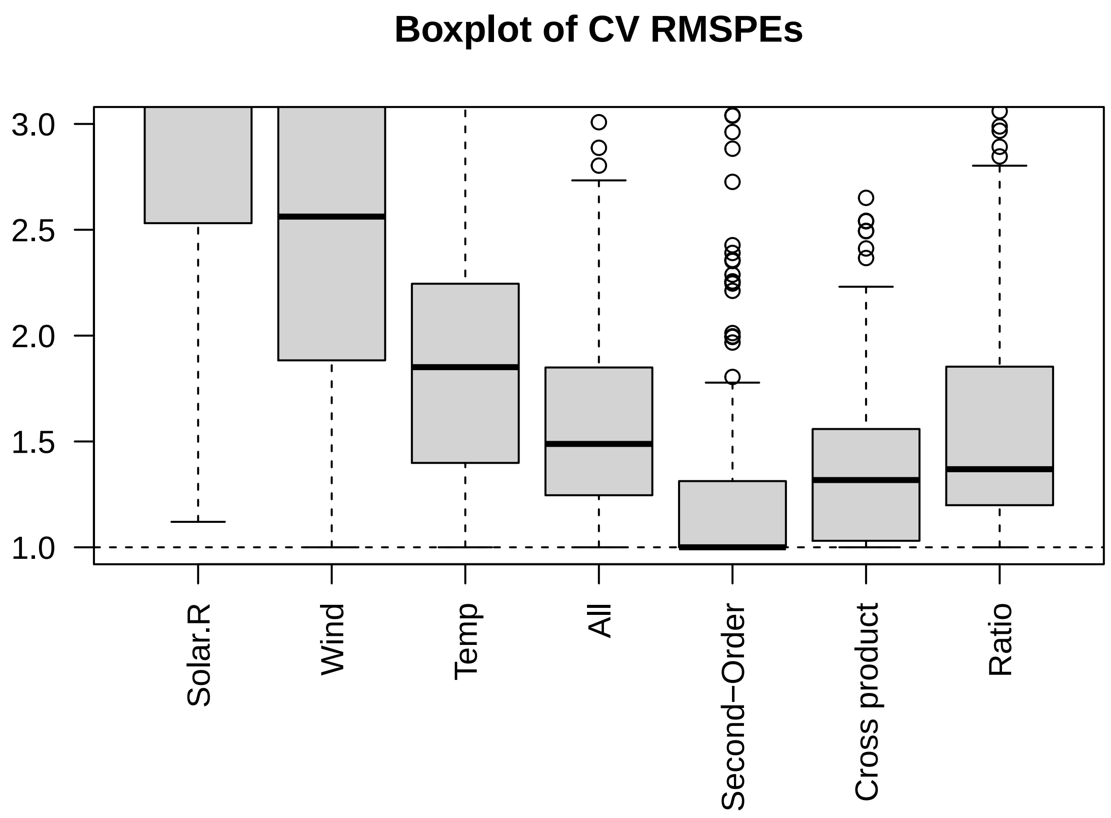
Because the MSPE from the validation data for cross product model (199) is less than the MSPE from the validation data for ratio model (218), the cross product model `fit.cp` wins this competition.

## 4.

### (a)

```r
### Create a function which gets predictions and calculates MSPEs
MSPE.lm = function(fit, data.valid, Y.valid) {
  pred = predict(fit, data.valid) # Predicted values on validation set
  errs = Y.valid - pred # Validation set residuals
  MSPE = mean(errs ^ 2)
  return(MSPE)
}

K = 10 # Number of CV folds

### Create function which constructs folds for CV
### n is the number of observations, K is the number of folds
get.folds = function(n, K) {
folds = floor((sample.int(n)-1)*K/n) + 1
return(folds)
}

M = 20 # Number of times to repeat CV

### Create a 3D array to store CV errors by model, then iteration,
### then fold. Set names of models.
all.CV.MSPEs = array(0, dim = c(7, M, K))
dimnames(all.CV.MSPEs)[[1]] = c("Solar.R", "Wind", "Temp", "All", "Second-Order",
                                "Cross product", "Ratio")

for (j in 1:M) {
  ### Get CV fold labels
  folds = get.folds(n, K)

  ### Perform cross-validation
  for (i in 1:K) {
    ### Get training and validation sets
    data.train = AQ[folds != i,]
    data.valid = AQ[folds == i,]

    ### Fit regression models
    fit.solar = lm(Ozone ~ Solar.R, data = data.train)
    fit.wind = lm(Ozone ~ Wind, data = data.train)
    fit.temp = lm(Ozone ~ Temp, data = data.train)
    fit.all = lm(Ozone ~ Temp + Wind + Solar.R, data = data.train)
    fit.2order = lm(Ozone ~ Temp*Wind + Temp*Solar.R + Wind*Solar.R
                    + I(Solar.R ^ 2) + I(Wind ^ 2) + I(Temp ^ 2), data = data.train)
    fit.cp = lm(Ozone ~ Temp + Wind + TWcp, data = data.train)
    fit.rat = lm(Ozone ~ Temp + Wind + TWrat, data = data.train)

    ### Calculate MSPEs
    ### Be careful with order of indices!!!
    all.CV.MSPEs["Solar.R", j, i] = MSPE.lm(fit.solar, data.valid, data.valid$Ozone)
    all.CV.MSPEs["Wind", j, i] = MSPE.lm(fit.wind, data.valid, data.valid$Ozone)
    all.CV.MSPEs["Temp", j, i] = MSPE.lm(fit.temp, data.valid, data.valid$Ozone)
    all.CV.MSPEs["All", j, i] = MSPE.lm(fit.all, data.valid, data.valid$Ozone)
    all.CV.MSPEs["Second-Order", j, i] = MSPE.lm(fit.2order, data.valid, data.valid$Ozone)
    all.CV.MSPEs["Cross product", j, i] = MSPE.lm(fit.cp, data.valid, data.valid$Ozone)
    all.CV.MSPEs["Ratio", j, i] = MSPE.lm(fit.rat, data.valid, data.valid$Ozone)
  }
}
```

```
### Combine all MSPEs for each model into a single list
### Note: as.numeric() converts its input into a 1D vector
data.CV.MSPEs = apply(all.CV.MSPEs, 1, as.numeric)

### Compute RMSPEs
data.CV.RMSPEs = apply(data.CV.MSPEs, 1, function(W){
best = min(W)
output = W / best
return(output)
})
data.CV.RMSPEs = t(data.CV.RMSPEs)

### Create boxplot
par(mar = c(6.5, 4.1, 4.1, 2.1))
boxplot(data.CV.RMSPEs, main = "Boxplot of CV RMSPEs", ylim = c(1,3), las = 2)
abline(h = 1, lty = 2)
```



**(b)**

The second-order model is still the best most often, but there are many cases where it is beaten. Our 2 new models win the competition sometimes, the All model seems to have similar performance with the Ratio model, and they all usually have better performance than the model that fit single variable.