Exercise 5 - Variable Selection

Multicollinearity and Variable Selection

Enter your name (REQUIRED)

Enter your completion date (REQUIRED)

Table of Contents

**Technical Issue:** Note that the YAML header in the template has these new parameters:

editor\_options:  
chunk\_output\_type: console

These parameters will cause your output to display in the R console and the graphs to display in the Plot viewer window, instead of inline after each code chunk. This can be useful if you want to keep your code clean and be able to zoom your graphs in a separate zoom window. But if you prefer R Markdown to display your work **inline** like we did before, you can simply delete these parameters in the YAML header or change it to (the document will knit the same either way):

editor\_options:  
chunk\_output\_type: inline

## Preparation

Download the **Ex5\_VariableSelection\_YourLastName.Rmd** R Markdown file to your working directory, rename with your last name and follow the instructions below. When you finish, upload onto Blackboard the .Rmd file or the knitted file as a Word, HTML or PDF file. Knitting and formatting is worth up to **3 points** in this and all exercises.

**Formatting:** Please ensure that all your text narratives are fully visible (if I can’t see the text, I can’t grade it). Also, please ensure that your **Table of Contents** is visible and properly formatted. Also, please prepare your R Markdown file with a **professional appearance**, as you would for top management or an important client. Please, write all your interpretation narratives in the text area, outside of the R code chunks, with the appropriate formatting and businesslike appearance. **Note:** I write all my interpretation solutions inside of the R code chunk to suppress their display until I print the solution, but you don’t need to do this. I will read your submission as a report to a client or senior management. Anything unacceptable to this audience is unacceptable to me.

**Submission**: Submit your knitted homework document in Canvas. There is no need to submit the .Rmd file, just your knitted file.

## 1. Read the Data

1.1 Use the **read.table()** function to retrieve columns **3 to 10** from the data set **PizzaCal.csv**  from Canvas and store it in your R working folder for this class. Name your dataset **pizza**.

# Done for you, but please ensure your data set has the columns listed below  
pizza <- read.table("PizzaCal.csv", header = T, sep = ",")[, 3:10]  
head(pizza)

## import mois prot fat ash sodium carb cal  
## 1 0 47.17 22.29 21.30 4.08 0.74 5.16 302  
## 2 0 49.16 27.99 17.49 3.29 0.39 2.07 278  
## 3 1 30.49 21.28 41.65 4.82 1.64 1.76 467  
## 4 0 52.68 14.38 25.72 3.26 0.93 3.96 305  
## 5 0 33.05 7.34 15.78 1.34 0.42 42.49 341  
## 6 0 35.55 7.32 16.40 1.76 0.36 38.97 333

This dataset contains nutrition data on a sample of 300 slices of frozen pizza (100 grams each):

* **import**: binary variable, 0 if domestic, 1 if imported
* **mois**: moisture content (grams per slice)
* **prot**: amount of protein (grams per slice)
* **fat**: fat content (grams per slice)
* **ash**: ash content (grams per slice)
* **sodium**: amount of sodium (grams per slice)
* **carb**: amount of carbohydrates (grams per slice)
* **cal**: number of calories in the slice

## 2. OLS

2.1 Fit an OLS model to predict **cal** with all other variables as predictors. Store the results in an object named **fit.full**. Display the summary() results for this model.

fit.full <- lm(cal ~ . , data = pizza)  
summary(fit.full)

##   
## Call:  
## lm(formula = cal ~ ., data = pizza)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -3.7246 -0.3716 -0.0677 0.2065 25.5575   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 608.7980 127.3020 4.782 2.75e-06 \*\*\*  
## import 0.4670 0.3579 1.305 0.1930   
## mois -6.0868 1.2734 -4.780 2.78e-06 \*\*\*  
## prot -2.0888 1.2724 -1.642 0.1017   
## fat 2.9244 1.2736 2.296 0.0224 \*   
## ash -6.0664 1.3219 -4.589 6.62e-06 \*\*\*  
## sodium -0.5532 1.0002 -0.553 0.5806   
## carb -2.0881 1.2729 -1.640 0.1020   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.584 on 292 degrees of freedom  
## Multiple R-squared: 0.9994, Adjusted R-squared: 0.9993   
## F-statistic: 6.542e+04 on 7 and 292 DF, p-value: < 2.2e-16

## 3. Multicollinearity Testing

3.1 First, load the **{klaR}** library and obtain the **Condition Index (CI)** for this model using the cond.index() function. Feed the model **fit.full** into this function, along with the data = Pizza parameter. Store the results in an object named **ci**.

**Technical Note:**

This object is a vector with ratios of the square root of the eigenvalues of each the correlation matrix against the smallest eigenvalue.

**Technical note:** As you know, eigenvalues are sorted by size. In the cond.index() function, the eigenvalues are ordered from largest to smallest. If eigenvalues are relatively even, there is little linear dependency or multicollinearity among the predictors. If the eigenvalues vary widely, there are substantial linear dependecies or multicollinearity in the data. The cond.index() function (like most CI functions) sorts the eigenvalues from smallest to largest. The CI is computed as the square root of the ration of the largest eigenvalue to the smallest. The larger this number, the larger the difference among eigenvalues, the more multicollinear the data. Most CI functions will compute the condition number for each eigenvector, as the square root of its eigen value to the smallest eigenvalue. But the only one that matters is the largest ratio, which is the definition of a CI.

Please display the full **ci** vector so that you can take a look at it, and then extract the **CI** , that is the last value, from the **ci** vector and display it. **Tip:** Since the function length() counts the number of elements in a vector, ci[length(ci)] will extract the last, that is the largest of these ratios from the **ci** vector, which is the **CI** multicollinearity statistic we are looking for.

library(klaR)  
ci <- cond.index(fit.full, data = pizza)  
ci

## [1] 1.000000 2.660398 3.224944 7.591463 15.584568 20.574711  
## [7] 34.274430 4046.578740

Then load the **{car}** library library to compute the Variance Inflation Factors (**VIF’s**). Obtain the VIFs\*\* for the predictors in the model, by feeding **fit.full** into the vif() function.

library(car)  
vif(fit.full)

## import mois prot fat ash sodium   
## 3.175904 17637.546103 7989.535468 15575.156318 335.803636 16.354718   
## carb   
## 62784.245725

3.2 Interpretation. Are there problems with multicollinearity? Please briefly explain in 1 or 2 lines why or why not. Also, which predictors are the top 2 contributors to multicollinearity in this model and why?

**Answer:**

Yes, there are problems with multicollinearity because the CI (4046.58) is higher than 50. Additionally, all the VIFs are above 10 apart from one indicating that predictors are linearly related.

The top 2 contributors among the predictors are carb and mois which have VIF of 62,784 and 17,637 respectively.

**3.3** Suggestions. Do you have any suggestions on how to resolve this multicollinearity issue? Provide one sound suggestion

\*\* Answer:\*\*

A possible suggestion would be explore some other better models through variable selection. Also, we can get more data which sometimes lowers the multicollinearity but the CI in this case is way too high that this may not be the case for this dataset.

## 4. Stepwise Variable Selection

Let’s select the best set of variables using **stepwise** variable selection to predict calories in the **pizza**  data set.

4.1 Suppose that your manager has told you that clients always inquire about **fat** and **carbs**, so she wants you to include these two predictors in the model, regardless of significance. So, rather than running a stepwise method between the null and full models, we will run a stepwise selection with a scope from a small model (with only fat and carbs) to the full model with all predictors.

So, first, fit the **small** model and name it **fit.small**, predicting calories (**cal**), with only **fat** and **carb** as predictors, which will be the lower boundary of the stepwise scope. Then display the summary() of this model and take a quick look at the results.

fit.small <- lm(cal ~ fat + carb, data = pizza)  
summary(fit.small)

##   
## Call:  
## lm(formula = cal ~ fat + carb, data = pizza)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -25.474 -10.727 1.582 8.751 32.847   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 87.50026 3.37217 25.95 <2e-16 \*\*\*  
## fat 8.77440 0.11361 77.23 <2e-16 \*\*\*  
## carb 2.71587 0.05656 48.02 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 13.54 on 297 degrees of freedom  
## Multiple R-squared: 0.9526, Adjusted R-squared: 0.9523   
## F-statistic: 2984 on 2 and 297 DF, p-value: < 2.2e-16

As you can see, both predictors are significant. Because of the high collinearity among predictors, we suspect that this **fit.small** model is biased, but we also know that the **fit.full** model suffers from severe multicollinearity. We could test **fit.small** for multicollinearity too, but instead, let’s go through the **Stepwise** variable selection model and test the multicollinearity of the final resulting model.

The **Stepwise** approach requires a **starting model** as the first parameter. We can start either with fit.small (which has a lot more bias) or fit.full (which has a lot more variance). You may get similar results with either, but it is generally better to start with the least biased model (i.e., fit.large) because the coefficients will be less biased. For this exercise, let’s run stepwise starting with **fit.full**.

That is, let’s enter **fit.full** as the first argument in the step() function.

The next parameter is scope = which provides the range of models to test. You can provide a list of models to try, if you wish. But in this exercise, you will explore a range of models. So, we need to feed the lower = and upper = parameters to define the lower and upper boundaries of the range of models to test. Since our starting model is **fit.full**, the upper bound model is upper = fit.full. The lower bound model is lower=fit.small, which contains the predictors your manager wants to see in the final model. Note that the step() function requires that we pass the scope as a list. In sum, your scope() function should look like this:

scope = list(lower = fit.small, upper = fit.full).

The next parameter in Stepwise is direction =. The default is "backward, which will do a backward selection and will **ONLY** eliminate non-significant predictors, but will **NOT** add any predictors. "forward will **ONLY** add predictors, but will not eliminate any predictors.

Let’s use direction = "stepwise", which will do both, remove and add. Since we are starting with **fit.full** there is no option but to go backwards, eliminating non-significant predictors. But if one of the predictors removed were to become significant if included, it will be added back at some point.

**Technical Note:** the default test when comparing models in step() is the **AIC**, which is an adjusted deviance measure, with little interpretation. For our purposes, it is better to use a standard statistical test like the **ANOVA F-test**. To change the default test to an **F-Test**, include the test = "F" parameter. Note, this is NOT F for false, but “F” in quotes as the letter F, which is the name of the test. This test will use **p < 0.15** as the default criterion to add and remove variables. Any included variable with a p >= 0.15 will be removed from the model. Any excluded variable that would have a significance of p < 0.15 if included in the model will be added back.

Save the resulting step() object with the name **fit.step.15p**. Then display it using the summary() function.

fit.step.15p <- step(fit.full,  
 scope = list(lower=fit.small,  
 upper=fit.full),  
 direction = "both",  
 test = "F")

## Start: AIC=283.8  
## cal ~ import + mois + prot + fat + ash + sodium + carb  
##   
## Df Sum of Sq RSS AIC F value Pr(>F)   
## - sodium 1 0.767 733.26 282.12 0.3059 0.5806   
## - import 1 4.270 736.76 283.55 1.7022 0.1930   
## <none> 732.49 283.80   
## - prot 1 6.761 739.25 284.56 2.6950 0.1017   
## - ash 1 52.829 785.32 302.69 21.0595 6.620e-06 \*\*\*  
## - mois 1 57.318 789.81 304.40 22.8490 2.781e-06 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Step: AIC=282.12  
## cal ~ import + mois + prot + fat + ash + carb  
##   
## Df Sum of Sq RSS AIC F value Pr(>F)   
## - import 1 3.767 737.03 281.65 1.5051 0.22087   
## <none> 733.26 282.12   
## - prot 1 6.906 740.17 282.93 2.7597 0.09774 .   
## + sodium 1 0.767 732.49 283.80 0.3059 0.58064   
## - mois 1 58.154 791.41 303.01 23.2375 2.303e-06 \*\*\*  
## - ash 1 58.199 791.46 303.03 23.2555 2.283e-06 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Step: AIC=281.65  
## cal ~ mois + prot + fat + ash + carb  
##   
## Df Sum of Sq RSS AIC F value Pr(>F)   
## <none> 737.03 281.65   
## + import 1 3.767 733.26 282.12 1.5051 0.22087   
## - prot 1 7.296 744.32 282.61 2.9104 0.08907 .   
## + sodium 1 0.264 736.76 283.55 0.1049 0.74622   
## - ash 1 58.904 795.93 302.72 23.4968 2.029e-06 \*\*\*  
## - mois 1 59.690 796.72 303.01 23.8105 1.745e-06 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

**Technical Note:** Notice two things in the output above:

1. All predictors retained in the model have a significance of p < 0.15. As we know, significant predictors are the ones with p < 0.05. The p < 0.15 is simply a significance criteria for which variables to retain in the model. In other words, predictors with significance levels at p >= 0.15 are dropped from the model.
2. fat and carb are retained in the model, and would be retained even if their p-values were >= 0.15. This is so because it is in the lower = parameter, as your manager wanted.

**Another Technical Note:**

p < 0.15 is a good criterion for variable inclusion and removal. Of course, we are interested in significant predictors at the p < 0.05 level, but it is important to retain marginally significant predictors as control variables. If you want a larger model, you can change the selection threshold to p < 0.20 or even higher. If you want a smaller model, you can use a more restrictive threshod like p < 0.10 or smaller. You can control this with a parameter k =. **k** is the **Chi-Square** equivalent to the p-value. The default in step() is **k = 2**, which is approximately equivalent to **p < 0.15**.

4.2 Let’s change the addition/removal significance value from p < 0.15 to p < 0.05. First, we need to find out the Chi-Square equivalent to p < 0.05. To do this, enter qchisq(0.05, 1, lower.tail = F). Save this result in a variable named **kval** and display its result. You should get **k = 3.84**. Then set the **k** parameter to **3.84** at the end of the scope() function (k = 3.84). Better yet, use k = kval, so that your script self-adjusts if you use a different value for **kval**. The re-fit **fit.step** but this time save the model as **fit.step.05p**

summary(fit.step.15p)

##   
## Call:  
## lm(formula = cal ~ mois + prot + fat + ash + carb, data = pizza)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -3.5961 -0.3576 -0.0795 0.1952 25.7092   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 618.801 126.986 4.873 1.80e-06 \*\*\*  
## mois -6.196 1.270 -4.880 1.75e-06 \*\*\*  
## prot -2.168 1.271 -1.706 0.0891 .   
## fat 2.822 1.269 2.223 0.0270 \*   
## ash -6.258 1.291 -4.847 2.03e-06 \*\*\*  
## carb -2.181 1.270 -1.717 0.0870 .   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.583 on 294 degrees of freedom  
## Multiple R-squared: 0.9994, Adjusted R-squared: 0.9993   
## F-statistic: 9.165e+04 on 5 and 294 DF, p-value: < 2.2e-16

4.3 Answer briefly: what is the difference in the final model when using p < 0.05 rather than p < 0.15 as the inclusion/removal criterion?

**Answer:**

p < 0.05 produced a much smaller model, which has four predictors because it is more strict in its criteria for inclusion of predictors while p < 0.15 produced a larger model with five predictors.

## 5. Re-testing for Multicollinearity Testing

5.1 Now that we have 2 reduced models identified by **stepwise** regression, compute the CI (cond.index() the parameter data = Pizza) and VIF (vif()) for each of the two models.

kval <- qchisq(0.05, 1, lower.tail = F)  
kval

## [1] 3.841459

5.2 What is your general conclusion from this analysis? Does any model solve the multicollinearity issue? **Technical tip**: the goal is to find the largest possible model without severe multicollinearity.

**Answer:**

In fit.step.15p, there is still high multi-collinearity (CI = 3,628.8) above 50. Additionally, all VIFs are also above 10. The multi-collinearity is reduced in fit.step.05p (CI = 71.7) even though it is still greater than 50.

# Done for you  
  
fit.better <- lm(cal ~ mois + fat + ash, data = pizza)  
cond.index(fit.better, data = pizza)

## [1] 1.000000 4.663655 8.495943 19.987587

vif(fit.better)

## mois fat ash   
## 1.853456 4.614729 4.819130

summary(fit.better)

##   
## Call:  
## lm(formula = cal ~ mois + fat + ash, data = pizza)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -4.4423 -0.3469 -0.0701 0.2075 25.7179   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 400.81798 0.59090 678.32 <2e-16 \*\*\*  
## mois -4.01628 0.01308 -307.17 <2e-16 \*\*\*  
## fat 4.99667 0.02196 227.55 <2e-16 \*\*\*  
## ash -3.99510 0.15863 -25.19 <2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.586 on 296 degrees of freedom  
## Multiple R-squared: 0.9994, Adjusted R-squared: 0.9993   
## F-statistic: 1.521e+05 on 3 and 296 DF, p-value: < 2.2e-16

# This final model only has significant predictors and no severe multi-collinearity. If your manager insists in keeping carb in the model, you need to explain that the model will have more than tolerable variance, due to multi-collinearity. But you can still use it for interpretation. But if you need to make accurate predictions, I would use the fit.better model.