Maths 2 : Devoir Maison de statistiques appliquées

Marco Freire Clément Legrand-Duchesne

25 mars 2018

Résumé

Mots-clefs :
Classification ACM :

Plan

1 Description du jeu de données

Le jeu de données fournis contient les mesures de la qualité des programmes couramment utilisés par les biologistes. Seize programmes sont ainsi examinés, selon des critères tels que le nombre de lignes ou de blocs dupliqués, le nombre et le type de warning lors de la compilation ou encore le statut donné par valgrind sur la gestion mémoire.

Un premier graphique nous permet de compter le nombre de programme écrit dans chaque langage du jeu de donnée (figure 1).

Nous avons ensuite décider de représenter le nombre de lignes de codes par programme (figure 2). Afin que la représentation soit plus lisible, nous avons utilisé la commande order pour ordonner les colonnes par ordres décroissants de valeurs. De plus, nous avons affiché les noms des programmes en vertical, afin d'améliorer la lisibilité (grâce à l'argument las = 2 de barplot).

De même, nous avons créer un histogrammes contenant le nombre de blocs dupliqués par programme (figure 3). Il est nécessaire de faire attention à ne pas prendre en compte les programmes pour lesquels cette valeur n'est pas renseignée (c'est le cas de *FDPPDIV* par exemple).

Enfin, nous avons généré un dernier graphique affichant le nombre de programmes par domaine (figure 4).

2 Pourcentage de lignes de code dupliquées

Afin de calculer le pourcentage de ligne dupliquées, nous avons créé une nouvelle table, contenant les noms des programmes, leurs dommaines, les langages dans lesquels ils ont été écrits et leurs nombres de lignes dupliquées (figure 5) ainsi que leurs nombres de lignes totales. Nous avons enlevé de cette nouvelle table de données les programmes pour ayant un champs non renseigné (à l'aide de la commande *na.omit*). Nous avons ensuite rajouté une colonne à celle ci contenant les pourcentages de lignes de codes dupliquées, avant de créer un histogrammes de ces pourcentages pour chaque programme, ordonné par ordre décroissant (figure 6).

3 Comparaison du nombre de lignes dupliquées en C et en C++

Pour effectuer un comparaison sur le nombre de lignes de codes des programmes C et C++, il nous faut faire un test de Mann-Whitney (grâce à la fonction *wilcox.test* en R. Pour que ce test soit valide, nous sommes obligés de supposer que les deux échantillons considérés suivent une loi normale, de variance identique. De plus, les mesures sont bien idépendantes les unes des autres.

Nous faisons pour hypothèse nulle "Il n'y a pas de différences significatives du nombre de lignes de code dupliquées en fonction du langage de programmation utilisé.". La p-value retournée par le test est de 0.94, au risque d'erreur 5%, nous ne pouvons donc pas rejeter l'hypothèse nulle. Nous en concluons que le nombre de lignes de codes duppliquées et le langage de programmation sont décorrélés.

Nous noterons que la longueur des codes biaise l'interprétation de la mesure : en effet,

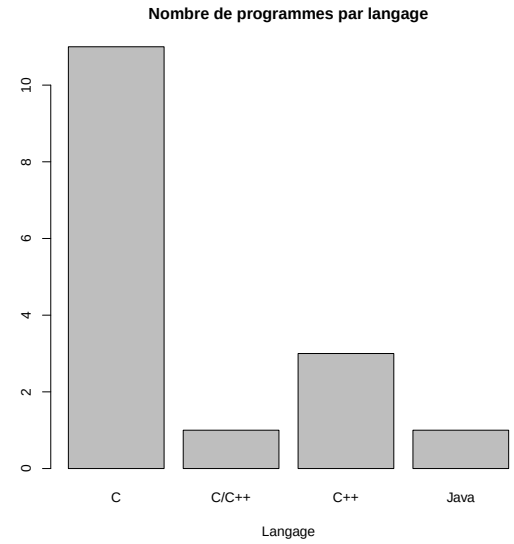

FIGURE 1

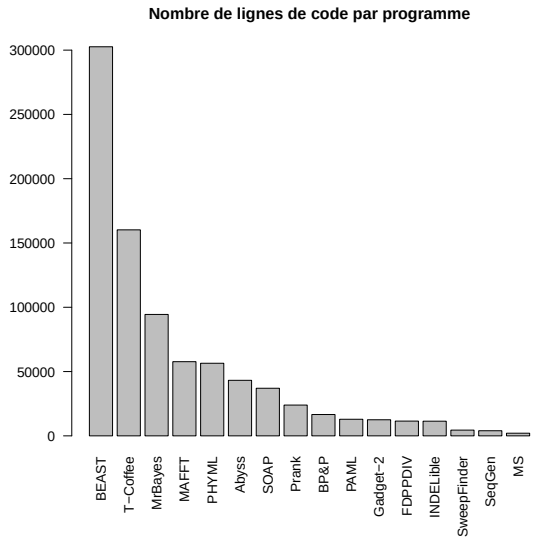

FIGURE 2

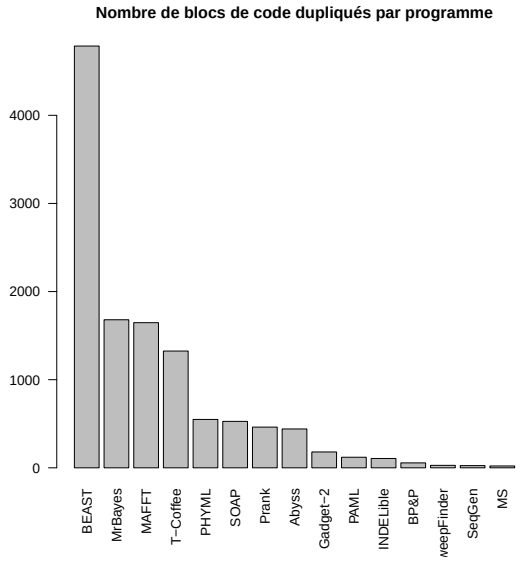

FIGURE 3

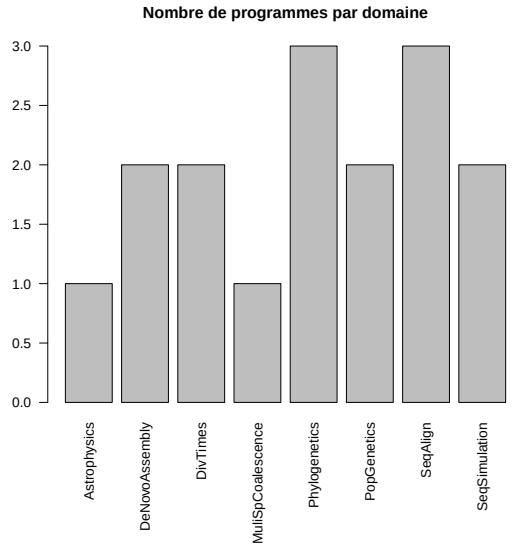

FIGURE 4

plus un code est long, plus le nombre de lignes duppliquées a de chance d'être élevé, aussi, un grand nombre de lignes de codes duppliquées ne reflète pas une mauvaise technique de programmation, tout dépend de la taille du code ! Nous avons donc aussi comparé le pourcentage des lignes de codes écrits en C et en C++.

L'hypothèse nulle devient "Il n'y a pas de différences significatives du pourcentage de lignes de code dupliquées en fonction du langage de programmation utilisé.". La p-value retournée par le test est de 0.41, au risque d'erreur 5%, nous ne pouvons donc pas rejeter l'hypothèse nulle. Nous en concluons que le langage de programmation et le pourcentage de lignes de code dupliquées sont décorréllés.

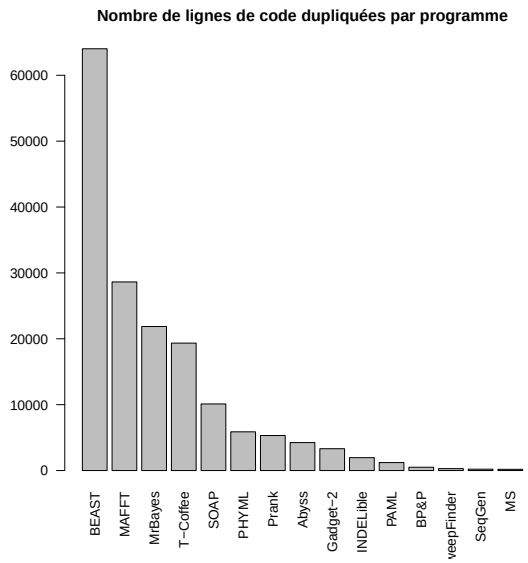

FIGURE 5

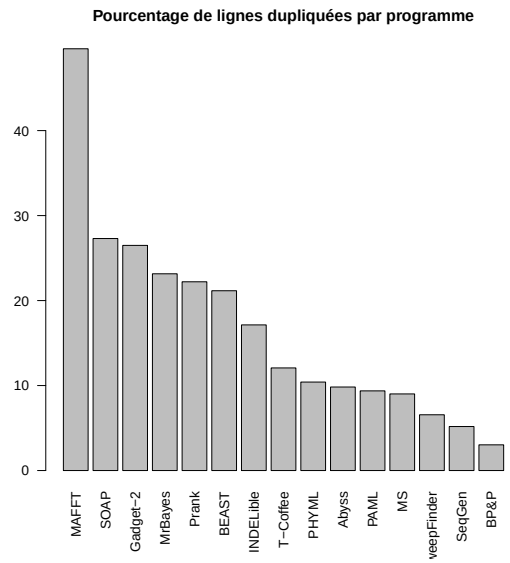

FIGURE 6

4 Intervalle de confiance du taux de lignes duppliquées

L'intervalle de confiance à 90 % du taux de lignes dupliqué est obtenu grâce à la fonction *t.test*. Il vaut [12.22694 ; 27.40435] pour les programmes écrites en C++ et [1.264158 ; 21.246497] pour les autres. La fonction *t.test* renvoie aussi les moyennes des échantillons.

5 Lien entre la longueur du code et le nombre de warning à la compilation

Nous vouons explorer l'hypothèse selon laquelle les codes longs comporteraient plus de warnings à la compilation que les courts. Nous avons commencé a séparer notre table en deux, l'une pour les programmes au nombre de lignes de codes inférieur à la médiane et l'autre pour les autres programmes.

Nous avons ensuite effectué trois test de Mann Whitney, un pour chaque type de warning (Clang, MinorWarning de gcc, MajorWarning de gcc), avec pour hypothèse nulle "Il n'y a pas de différence significative du nombre de warning à la compilation entre les programmes de longueurs supérieure et inférieure à la médiane.". Nous avons pour cela encore une fois supposé que les échantillons considérés suivaient une loi normale de variance identique. Lors du test, nous avons passé *alternative = "less"* en argument à la fonction *wilcox.test*, cela correspond au fait que nous nous attendons à ce que le nombre de warning soit supérieur pour pour les programmes long.

Pour les Warning Clang, la p-value retournée par le test est de 0.0006216, au risque d'erreur 5%, nous pouvons donc rejeter l'hypothèse nulle. Nous en concluons que le nombre de warnings Clang est supérieur pour les programmes longs.

Pour les Warning gcc, la p-value retournée par le test est de 0.6039 et 0.05896 pour les warning mineurs et majeurs respéctivement. Au risque d'erreur 5%, nous ne pouvons donc

pas rejeter l'hypothèse nulle. Nous en concluons que le nombre de warning gcc n'est pas sensiblement supérieur pour les programmes longs.

Le test de Mann Whitney étant un test de comparaison de moyennes, nous avons voulu approfondir lien entre le nombre de warnings Clang et la taille des programmes. Pour cela, nous avons décider de faire un test de corrélation entre ces deux grandeurs. Nous avons pour cela effectué un test paramétrique de corrélation de Pearson avec la fonction *cor.test*. Ce test nécéssite que les données suivent une loi normale, cette condition étant ici discutable, nous avons aussi effectué un test non-paramétrique : le test de corrélation de Spearman, disponible avec la même fonction.

Le test de Pearson renvoie une p-value de 0.03095, et le test de Spearman de 0.001907. Nous pouvons donc affirmer au risque d'erreur 5 % que le nombre de lignes de codes et le nombre de warnings Clang sont corrélés.

Nous avons donc généré un graphique représentant le nombre de warnings Clang en fonction du nombre de lignes de code, sur lequel nous avons tracé une régression linéaire (figure 7).

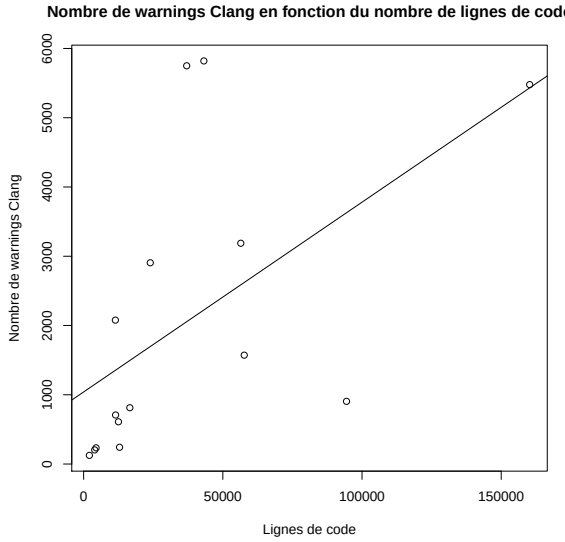

FIGURE 7

6 Questions libres

6.1 Lien entre le nombre et le type de warning à la compilation et la qualité de la gestion mémoire.

Nous avons choisi de nous demander si il existe un lien entre le nombre de warning de chaque type renvoyé pendant la compilation et la qualité de la gestion mémoire attribuée par valgrind.

Nous pour cela séparé notre jeu de données en deux : les programmes ayant une gestion mémoire décrite comme clean par valgrind, et les autres. Nous avons ensuite effectué trois test de Mann Whitney, un pour chaque type de warning (Clang, MinorWarning de gcc, MajorWar-

ning de gcc), avec pour hypothèse nulle “Il n’y a pas de différence significative du nombre de warning à la compilation entre les programmes en fonction de la qualité de leur gestion mémoire.”.

Pour les Major Warning, la p-value retournée par le test est de 0.02165, au risque d’erreur 5%, nous pouvons donc rejeter l’hypothèse nulle. Nous en concluons que le nombre de warning majeur gcc est supérieur pour les programmes ayant une gestion mémoire non propre (figure 8).

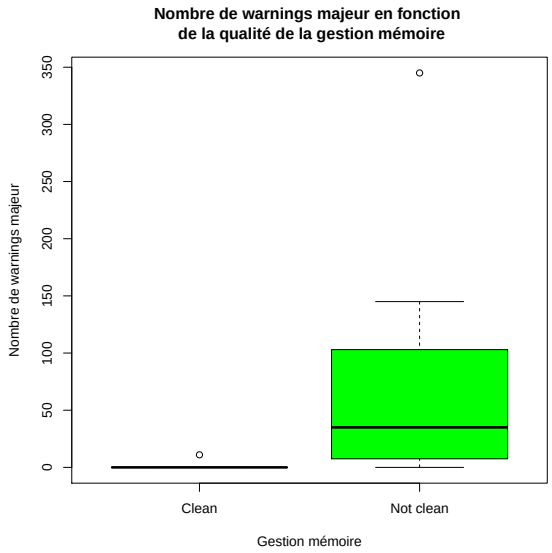

FIGURE 8

Pour les Warning mineurs et Clang respectivement, la p-value retournée par le test est de 0.1645 et 0.7348. Au risque d’erreur 5%, nous ne pouvons donc pas rejeter l’hypothèse nulle. Nous en concluons que le nombre de warning mineur gcc et clang n’est pas sensiblement inférieur pour les programmes propres.

6.2 Autre question

Enfin, nous avons cherché à déterminer si les programmes un domaine en particulier comportait une proportion plus importante de lignes dupliquées. Nous avons pour cela effectué un test de Kruskal-Wallis sur le pourcentage de lignes duppliquées par programme en fonction du domaine.

L’hypothèse nulle est “Il n’y a pas de différences significatives du pourcentage de lignes de code dupliquées en fonction du domaine considéré.” La p-value retournée par le test est de 0.2814, nous ne pouvons donc pas rejeter l’hypothèse nulle au risque de 5 %. Nous en concluons que le domaine et le pourcentage de lignes de codes dupliquées ne sont pas liés.

Annexe

```
setwd("~/marco/Documents/Cours/sem2/Stat/TPs/Eval/urban-funicular")
sink("rapport/tests/tests.txt", append = TRUE)
tab = read.table("programs.txt", header = TRUE, sep = "\t" )
head(tab)
summary(tab)

################################################################################

# Question 1

## Nombre de programmes par langage
pdf('rapport/figures/prog_lang.pdf')
barplot(table(tab$Language),
        main = "Nombre de programmes par langage",
        xlab = "Langage")
dev.off()

## Nombre de programmes par domaine
pdf('rapport/figures/prog_dom.pdf')
op = par(mar = c(9, 4, 4, 2) + 0.1)
barplot(table(tab$Domain),
        main = "Nombre de programmes par domaine",
        las = 2)
dev.off()

## Nombre de lignes de code par programme, ordonné par ordre décroissant
pdf('rapport/figures/lin_prog.pdf')
op = par(mar = c(7, 4, 4, 2) + 0.1)
tab_lin = tab[order(tab$LinesOfCode, decreasing = TRUE),]
barplot(tab_lin$LinesOfCode,
        names.arg = tab_lin$Code,
        las = 2,
        main = "Nombre de lignes de code par programme")
dev.off()

## Nombre de lignes dupliquées par programme, ordonné par ordre décroissant
pdf('rapport/figures/dlin_prog.pdf')
tab_dlines = tab[order(tab$DuplicatedLines, decreasing = TRUE),]
tab_dlines_noNA = tab_dlines[!is.na(tab_dlines$DuplicatedLines),]
barplot(tab_dlines_noNA$DuplicatedLines,
        names.arg = tab_dlines_noNA$Code,
        las = 2,
        main = 'Nombre de lignes de code dupliquées par programme')
dev.off()

## Nombre de blocs dupliquées par programme, ordonné par ordre décroissant
pdf('rapport/figures/dbl_prog.pdf')
```

```
tab_dblocks = tab[order(tab$DuplicatedBlocks, decreasing = TRUE),]
tab_dblocks_noNA = tab_dblocks[!is.na(tab_dblocks$DuplicatedBlocks),]
barplot(tab_dblocks_noNA$DuplicatedBlocks,
        names.arg = tab_dblocks_noNA$Code,
        las = 2,
        main = 'Nombre de blocs de code dupliqués par programme')
dev.off()

################################################################################

# Question 2

tabP = na.omit(tab[, c("Code", "Domain", "Language", "DuplicatedLines", "LinesOfCode")])
tabP$DupLin_per = tabP$DuplicatedLines / tabP$LinesOfCode * 100
tabP = tabP[order(tabP$DupLin_per, decreasing = TRUE),]

## Pourcentage de lignes dupliquées par programme
pdf('rapport/figures/pdlin_prog.pdf')
barplot(tabP$DupLin_per,
        names.arg = tabP$Code,
        las = 2,
        main = "Pourcentage de lignes dupliquées par programme")
dev.off()

################################################################################

# Question 3

tabP_C = tabP[tab$Language == "C",]
tabP_CPP = tabP[tab$Language == "C++",]

wilcox.test(tabP_C$DuplicatedLines, tabP_CPP$DuplicatedLines)
wilcox.test(tabP_C$DupLin_per, tabP_CPP$DupLin_per)

pdf('rapport/figures/pdlin_lang.pdf')
boxplot(tabP_C$DupLin_per, tabP_CPP$DupLin_per,
        na.rm = TRUE, col = c("blue", "green"),
        xlab = "Langage",
        ylab = "Pourcetage de lignes dupliquées",
        names = c("C", "C++"),
        main = "Pourcentage de lignes dupliquées en fonction \n du langage de programmation")
dev.off()
################################################################################

# Question 4

tabP_CO = tabP[tab$Language %in% c("C++", "C/C++"),]

t.test(tabP_C$DupLin_per, conf.level = 0.9)
```

```r
t.test(tabP_CO$DupLin_per, conf.level = 0.9)

################################################################################

# Question 5

tab_Clang = tab[!is.na(tab$ClangWarning), ]

tab_short = tab_Clang[tab_Clang$LinesOfCode <= median(tab_Clang$LinesOfCode),]
tab_long = tab_Clang[tab_Clang$LinesOfCode > median(tab_Clang$LinesOfCode),]

wilcox.test(tab_short$ClangWarning, tab_long$ClangWarning, alternative = "less")
wilcox.test(tab_short$MinorWraning, tab_long$MinorWraning, alternative = "less")
wilcox.test(tab_short$MajorWarning, tab_long$MajorWarning, alternative = "less")

# ne suivent pas une loi normale
shapiro.test(tab_Clang$LinesOfCode)
shapiro.test(tab_Clang$ClangWarning[na.rm = TRUE])

# Test de corrélation paramétrique de Pearson
cor.test(tab_Clang$ClangWarning, tab_Clang$LinesOfCode)
# Test de corrélation non−paramétrique de Spearman
cor.test(tab_Clang$ClangWarning, tab_Clang$LinesOfCode, method = "spearman")

pdf('rapport/figures/clang_lin.pdf')
plot(tab_Clang$LinesOfCode, tab_Clang$ClangWarning,
     xlab = "Lignes␣de␣code", ylab = "Nombre␣de␣warnings␣Clang",
     main = "Nombre␣de␣warnings␣Clang␣en␣fonction␣du␣nombre␣de␣lignes␣de␣code")
mod = lm(tab_Clang$ClangWarning ~ tab_Clang$LinesOfCode)
abline(mod)
dev.off()

# Reset graphic parameters
op = par(mar = c(5, 4, 4, 2) + 0.1)

################################################################################

# Question 6

## Influence des evaluations de valgrind sur le nombre de warnings

tabW = na.omit(tab[, c("MajorWarning", "MinorWarning", "ClangWarning", "Valgrind")])

tabW_CL = tabW[tabW$Valgrind == "clean" | tabW$Valgrind == "leaks",]
Clean = tabW_CL[tabW_CL$Valgrind == "clean",]
NClean = tabW_CL[tabW_CL$Valgrind != "clean",]

# Comparaison des moyennes des programmes ayant une évaluation valgrind "CLEAN" et les autres
```

```
# > On peut affirmer qu'il existe une différence significative entre les
# > moyennes au risque de 5% pour les major warnings
wilcox.test(Clean$MajorWarning, NClean$MajorWarning, alternative = "less")
wilcox.test(Clean$MinorWarning, NClean$MinorWarning, alternative = "less")
wilcox.test(Clean$ClangWarning, NClean$ClangWarning, alternative = "less")

pdf('rapport/figures/MW_valgr.pdf')
boxplot(Clean$MajorWarning, NClean$MajorWarning,
        col = c("blue", "green"),
        xlab = "gestion␣mémoire",
        ylab = "Nombre␣de␣warning␣majeur",
        names = c("Clean", "Not␣clean"),
        main = "Nombre␣de␣warning␣majeur␣en␣fonction␣\n␣de␣la␣qualité␣de␣la␣gestion␣mémoire")
dev.off()

## Influence du domaine scientifique sur le pourcentage de lignes dupliquées

# Pas de différence des moyennes significative au risque de 5%
kruskal.test(tabP$DupLin_per ~ tabP$Domain)
```