DCC888 – Machine Learning in Static Analysis

Name: _____ ID: _____

1. This exercise refers to the data available in the following CSV file: `https://www.dcc.ufmg.br/~fernando/classes/dcc888/ementa/data_ml/branch_features.csv`. We can use the script below to apply a logistic classification model onto this data. This script will shuffle and draw a training set for the branch prediction problem 1,000 times. Each time, it will compute the accuracy of the logistic regression model based on an ensemble of six features: `Break`, `BackEdge`, `EqZero`, `IsZero`, `NoPDom` and `ToFun`. In the end, it will output the average accuracy accumulated over these 1,000 iterations:

```
01 data <- read.csv(branch_features.csv)
02 missing_factor <- 0.2
03 sum_accuracies <- 0
04 num_iterations <- 1000
05 for(i in 1:num_iterations) {
06   missing_data <- data[sample(nrow(data), nrow(data)*missing_factor),]
07   shuffled_data <- missing_data[sample(1:nrow(missing_data)), ]
08   num_samples <- nrow(shuffled_data)
09   train_size <- ceiling(num_samples*0.8)
10   train <- shuffled_data[1:train_size,]
11   test <- shuffled_data[(train_size+1):num_samples,]
12   model <- glm(Taken ~ Break + BackEdge + EqZero + IsZero + NoPDom + ToFun,
13                family=binomial(link='logit'), data=train)
14   results <- predict(model, test, type="response")
15   predictions <- ifelse(results > 0.5, 1, 0)
16   misses <- mean(predictions != test$Taken)
17   accuracy <- 1 - misses
18   sum_accuracies <- sum_accuracies + accuracy
19 }
20 sum_accuracies/num_iterations
```

   (a) The accuracy of the learning model depends on the amount of available data. To see how this statement is true, we shall remove part of the dataset, and then retrain the model with the partial data that remains available. The variable called `missing_factor` represents the percentage of the data that we will remove from the dataset before performing the classification. The more data we remove, the less accurate the model is likely to be. Try changing this variable to see how the accuracy changes. What is the average accuracy that you obtain for values in the set $\{0.05, 0.1, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40\}$?

   (b) We define the size of the training set in Line 09. Currently, we are using 80% of the data as the training set. Try varying this proportion. What is the average accuracy that you obtain if the size of the training set is $\{80\%, 70\%, 60\%, 50\%, 40\%\}$ of the total available data?

(c) The model that we use is defined in Line 12. This is a strictly linear model, meaning that there is no dependency between the explanatory variables, e.g.: `Break`, `BackEdge`, `EqZero`, `IsZero`, `NoPDom` and `ToFun`. Try to add a seventh explanatory factor to your model, formed by the product: `IsZero` × `EqZero`. You can add this extra component by editing the model in lines 12 and 13, e.g.: `Taken Break + ... + ToFun + IsZero * EqZero`. What is the average accuracy that you obtain in this case?

(d) It is likely that the accuracy of your model will increase, once you add the seventh explanatory factor `IsZero` × `EqZero` to it. Try to find a reasonable explanation on why these two features, `IsZero` and `EqZero` are interdependent.

2. This exercise refers to the data available in the following CSV file: `https://www.dcc.ufmg.br/~fernando/classes/dcc888/ementa/data_ml/dyn_count.csv`. This file contains data observed by analyzing 27,672 programs after they were optimized with either `opt -O1` or `opt -O2`. The following columns will be of interest to us:

staticO0: Number of different instructions visited during the execution of the program compiled with `clang -O0`.

dynamicO0: Number of instructions fetched during the execution of the program compiled with `clang -O0`.

staticO2: Number of different instructions visited during the execution of the program compiled with `clang -O2`.

dynamicO2: Number of instructions fetched during the execution of the program compiled with `clang -O2`.

These numbers are *real*: they have been produced via dynamic analysis, using, to this end, the `CFGGrind` tool (available at `https://github.com/rimsa/CFGgrind`), applied onto programs from the AnghaBench collection (available at `http://cuda.dcc.ufmg.br/angha/home`).

Many machine learning algorithms and techniques assume that data comes from a *normal* distribution. Such is the case, for instance, of Pearson's Correlation. In this exercise, we shall analyze if the speedup observed after optimizations follows a normal distribution. To help you out, Figure 1 shows a distribution of speedups produced by `clang -O2` over `clang -O0`.

(a) The histogram in Figure 1 has been cropped at a speedup of 6x, where speedup is defined as the ratio `dynamicO2/dynamicO0`. There are programs, among the 27,672 available samples, that present larger speedups. How many programs present a speedup equal or larger than 6.0x?

(b) What is the largest speedup in our dataset?

(c) Why is it possible to obtain such a large speedup due to compiler optimizations?

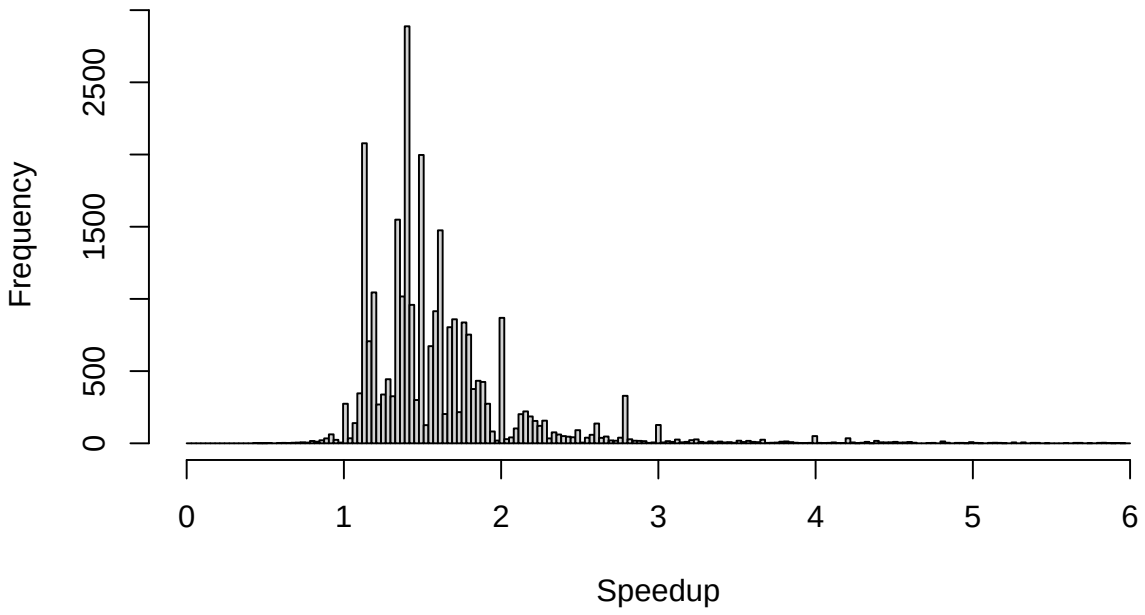

Figure 1: Histogram of speedups of O2 over O1 (`dynamicO2/dynamicO0`), cropped at 6x.

(d) Is there a theoretical limit to the speedup that can be obtained through compiler optimizations, or, for any speedup $s, s > 1.0$ can you design a program whose optimized version presents a speedup larger than s?

(e) The R standard library contains a function `shapiro.test` that applies the Shapiro –Wilk test to estimate the probability that data comes form a normal distribution. The higher Shapiro's p-value, the higher the probability that the data comes from a normal distribution. What is Shapiro's p-value for our dataset? Is this p-value an indication of normality?

(f) What is the average speedup that we obtain by analyzing the ratio `data$dynamicO0 /data$dynamicO2` in our dataset?

(g) The dataset contains also results for other optimization levels, e.g.: `dynamicO1` and `dynamicOz`. What is the average speedup that we observe at these different levels?