Parallelization of Searching Parameters for Support Vector Machine

Samiul Islam
Department of Data Science and Analytics
Ryerson University
Toronto, Canada
samiul.islam@ryerson.ca

Khaled Hyder
Department of Data Science and Analytics
Ryerson University
Toronto, Canada
mdkhaled.hyder@ryerson.ca

***Abstract*— Model regularization parameters in Support vector machines (SVM) play a vital role on models' efficiency as well as to boost up the models' predictive power. SVM has low tendency to over-fit which allows a better choice of free parameters. Tuning of parameters has been offered through many packages depending on various programing language platforms. Many of those packages offers sequential execution of tuning which led a time inefficient scenario. In this work, parallel execution of searching parameters has been examined and performance comparisons have been made with multiple core along with multiple data folds. Our result shows that higher number of processing cores performs better when data split increased to a certain level.**

Keywords—SVM, parameter searching, parallelization.

I. Introduction

Support vector machines (SVM) provides sinewy kernel methods for regression and classification. SVM able to produce splendid separating hyperplanes when trained optimally. The lineament of the training phase contingent upon on the quality and distribution of training data as well as on learning parameters. Learning parameters of a sample balanced dataset can be tuned to achieve better results on hyperplane separation. However, for an unbalanced dataset, this task become time consuming. In this paper, parallel execution of a tuning package has been evaluated to observe the effect of parallelization on multiple data folds.

II. Dataset Description

Experiment was carried out on a financial institute data set, focused on expenditure and default data [Greene 1992]. This data set contains 113444 observations with 14 attributes. The 'Cardhldr' attribute is a dummy variable, 1 if application for credit card accepted, 0 if not. The 'Default' attribute provides binary output, 1 if defaulted 0 if not (observed when Cardhldr = 1, 10,499 observations. The 'Age' attribute provides numerical data such as age in years plus twelfths of a year. The 'Adepcnt' shows the number of dependents + 1. The 'Acadmos' attribute privides the number of months living at current address. The 'Majordrg' attribute shows the number of major derogatory reports. The 'Minordrg' attribute provides the number of minor derogatory reports. The 'Ownrent' feature provides binary outcome, 1 if owns their home, 0 if rent. The 'Income' feature provides the monthly income (divided by 10,000). The 'Selfempl' attribute is set to 1 if self-employed, 0 if not. The 'Inc_per' attribute holds the number where income divided by number of dependents. The 'Exp_Inc' attribute provides the ratio of monthly credit card expenditure to yearly income. The 'Spending' feature provides the average monthly credit card expenditure. The 'Logspend' attribute provides the log of spending [Greene 1992].

III. Data Preprocessing

At first, null values have been removed from the dataset for data cleaning purpose. After data cleaning we have noticed the changes in data tendency of our eleven out of fourteen observation variables. The 1st attribute CARDHLDR's mean value changed to 1 from 0.789, which means we have removed all zeros from this attribute. In second attribute "DEFAULT" we can see the mean value raised to 0.09 from 0.07. The mean age gains a slight increase of 0.2 years. In "ACADMOS" the mean value increased slightly to 55.9. We have seen three decreases in features, these are "ADEPCNT", "MAJORDRG" and "MINORDRG". In all cases the mean value decreased slightly. The mean income raised 2606(total gain 96$) and there is sharp increase in "INCPER" which step up to 22581 from 21719. No changes in "SELFEMPL", "SPENDING" and "LOGSPEND" features have been observed.

IV. Experimental Settings

Experiment has been carried out on Intel core i7-5500U CPU with 4 available processing cores with a maximum processing speed of 2.40 GHz on each core, where the operating system is 64 bits. The “e1071” R package contains the tune function which try to find the SVM parameters sequentially. This parameter searching function has been parallelized among 4 available processors. The distribution was in 3 phases based on data split on K-Fold, where K=2,4 and 8. Each phase contains multiple settings on single core execution time, duel core execution time and quad core execution time with a variable cost ranging from 1 to 100. Time constrains have been observed from each phase for further analysis.

V. Results

A. *Before split:*

A test run has been performed before splitting the data where K=1, which reflects that time variation is almost identical for single core, 2 cores and 4 cores. The increase in performance occurs only with increasing cost values.

B. *After split:*

While train data has been split into K=2, the increase in performance among single core and 2 cores is noticeable. However, the performance of 2 cores and 4 cores are merely visible. With the data split K=4, the performance of single core, 2 cores are clearly separable, however the performance of 4 cores does not improve rapidly as compare to 2 cores. There is a rapid increase in performance between the cost value c= 10 to 40 has been observed and after c= 50 to 100 the performance improvement remains almost constant. With the data split K=8, the performance of single core, 2 cores and 4 cores are noticeably distinguishable. In this setting, a rapid improvement in performance for 4 cores has been observed when the c value shifted from 0 to 20. There is also a rapid increase in performance between the cost value c= 0 to 20 has been observed for 2 cores and 4 cores and there is a stable improvement in performance has been noticed after the cost reaches to 50. With the data split K=16, the performance of 2 cores and 4 cores are merely separable. More interestingly, it has been observed that for lower costs, 2 cores perform well than core when data split increases.

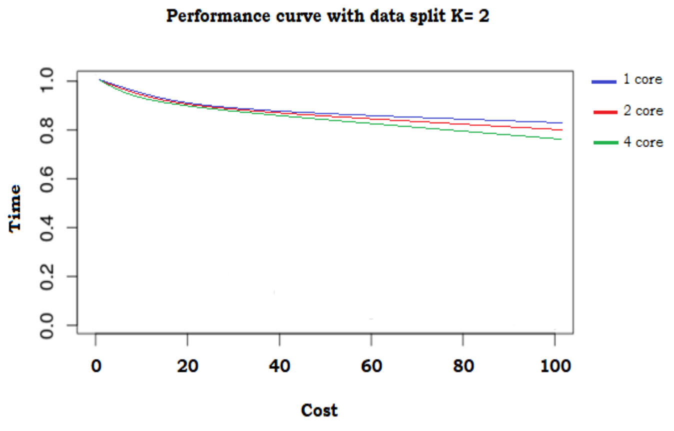

Fig. 1. Performance curve when K=2

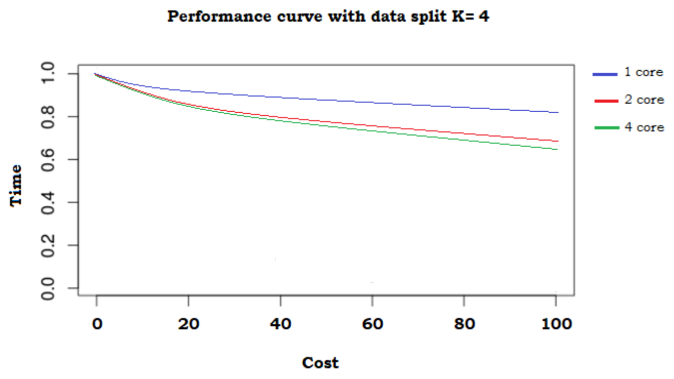

Fig. 2. Performance curve when K=4

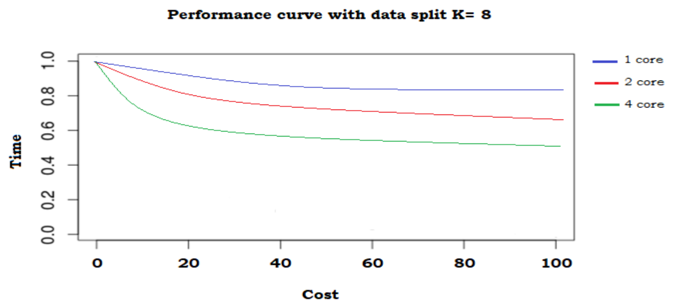

Fig. 3. Performance curve when K=8

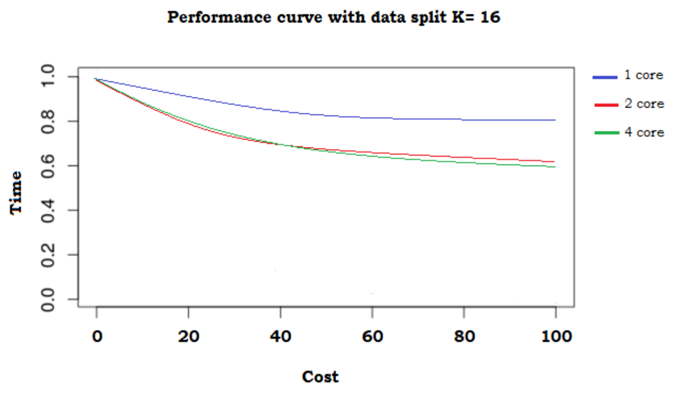

Fig. 4. Performance curve when K=16

VI. Conclusion

Experimental settings were not diverse enough to clearly reflect the performance gain, however the idea of this work is to have a glimpse on how parallelization of multiple split of a dataset can improve the performance by reducing the execution time. It has been observed that for lower costs, 2 cores perform well than 4 cores when data split increases at K≥16 for the given dataset.

References

Greene 1992 http://pages.stern.nyu.edu/~wgreene/Text /econometricanalysis.htm

Appendix:

```
#===Loading require libraries===#

library("parallel", lib.loc="C:/Program Files/R/R-3.4.2/library")
library("doParallel", lib.loc="~/R/win-library/3.4")
library("foreach", lib.loc="~/R/win-library/3.4")
library("iterators", lib.loc="~/R/win-library/3.4")
library("pROC", lib.loc="~/R/win-library/3.4")
library("caret", lib.loc="~/R/win-library/3.4")
library("e1071", lib.loc="~/R/win-library/3.4")


#===Initializing and assigning number of processing core/s===#

package <- c('foreach', 'doParallel')
lapply(package, require, character.only = T)
registerDoParallel(cores = 4)


#===Dataset loading and observations===#

Credit_data <- read.csv("E:/SAMIUL RYERSON/fall2017/Algorithm2017/Project/credit_count.csv")
View(Credit_data)
summary(Credit_data)


#===Data preprocessing===#
Clean_data <- Credit_data[Credit_data$CARDHLDR == 1, ]
View(Clean_data)
summary(Clean_data)
dt.hd <- paste("AGE + ACADMOS + ADEPCNT + MAJORDRG + MINORDRG + OWNRENT + INCOME + SELFEMPL
+ INCPER + EXP_INC")
fn1 <- as.formula(paste("as.factor(DEFAULT) ~ ", dt.hd))


#===Data split into the number of K-Folds===#

set.seed(666)
Clean_data$fold <- caret::createFolds(1:nrow(Clean_data), k = 8, list = FALSE)


#===List of Support Vector Machine Parameters===#

newcost <- c(10, 100)
newgamma <- c(1, 2)
parameters <- expand.grid(newcost = newcost, newgamma = newgamma)


#===Passing parameter values===#

Result1 <- foreach(m = 1:nrow(parameters), .combine = rbind) %do% {
cost <- parameters[m, ]$newcost
gamma <- parameters[m, ]$newgamma
#===Validation testing of K-Fold for a given K ===#

output <- foreach(n = 1:max(Clean_data$fold), .combine = rbind, .inorder = FALSE) %dopar% {
training <- Clean_data[Clean_data$fold != n, ]
testing <- Clean_data[Clean_data$fold == n, ]
modeling <- e1071::svm(fn1, data = training, type = "C-classification", kernel = "radial", newcost = cost, newgamma =
gamma, probability = TRUE)
predicting <- predict(modeling, testing, decision.values = TRUE, probability = TRUE)
data.frame(x = testing$DEFAULT, prob = attributes(predicting)$probabilities[, 2])
  }


#=== Performance evaluation ===#
time1 <- Sys.time() # To fetch the starting time
Perf_curve <- pROC::roc(as.factor(output$x), output$prob)
data.frame(parameters[m, ], Perf_curve = Perf_curve$auc[1])
}
execution_time <- Sys.time() - time1 # execution time calculation
print(execution_time) # shows the execution time
```