KNN

Jan Alexander Jensen

2020/4/26

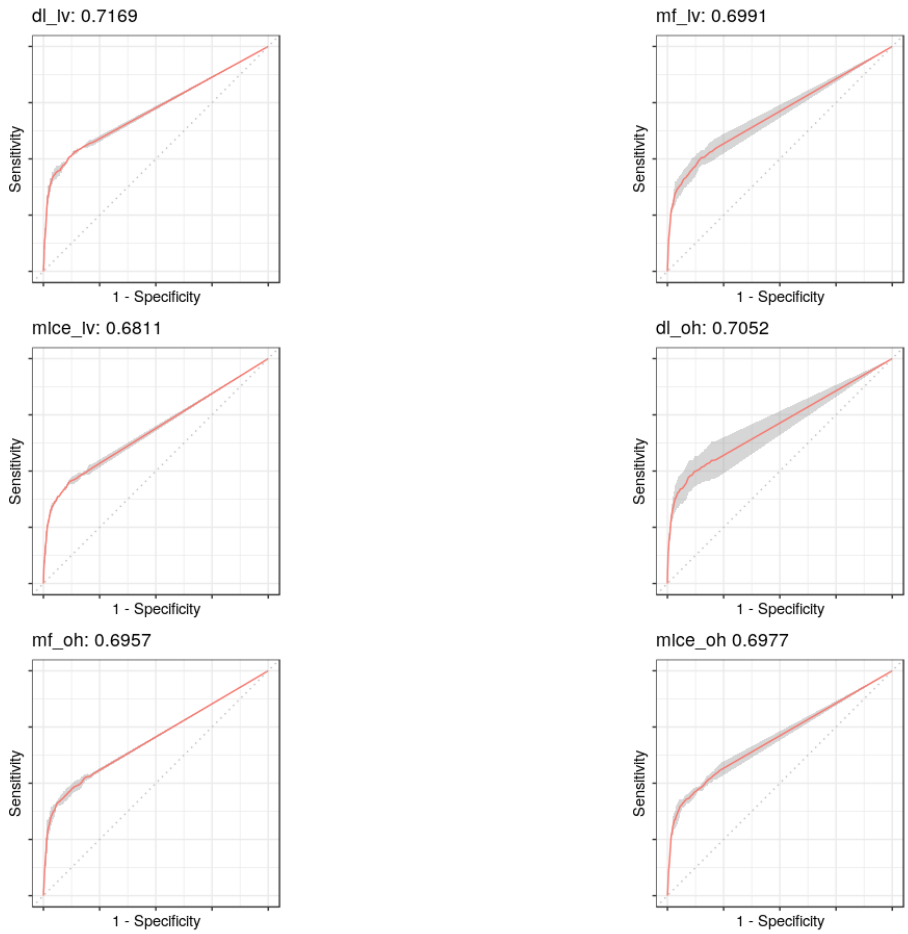
KNN

KNN (k-nearest neighbors) is a method used for classification. In simple words, to classify one specific data point, it takes k neighbors with the shortest distance. Furthermore, based on how the neighbors are classified, we will assign the new input to the most popular category in the k neighbors. Here the number of k and the method of calculating the distances between data points are crucial. We will first evaluate how KNN performs between different data, including different encoding, and different missing value handling. Then we will explore different methods to get better results. This will be described in detail in the next section.

```
library(mlr3)
library(mlr3learners)
library(mlr3tuning)
library(mlr3pipelines)
library(tidyverse)
library(ggplot2)
library(gridExtra) # for merging plots in one

# suppress package making warning by start up in train
# Warning: "package 'kknn' was built under R version 3.6.3"
suppressPackageStartupMessages(library(kknn))

# read data with different encoding
dl_iv_data <- read.csv2("credit_card_prediction/iv_data/dl_iv_data.csv") %>% mutate(y = as.factor(y))
mf_iv_data <- read.csv2("credit_card_prediction/iv_data/mf_iv_data.csv") %>% mutate(y = as.factor(y))
mice_iv_data <- read.csv2("credit_card_prediction/iv_data/mice_iv_data.csv") %>% mutate(y = as.factor(y

dl_oh_data <- read.csv("credit_card_prediction/oh_data/dl_oh_data.csv") %>% mutate(y = as.factor(y))
mf_oh_data <- read.csv("credit_card_prediction/oh_data/mf_oh_data.csv") %>% mutate(y = as.factor(y))
mice_oh_data <- read.csv("credit_card_prediction/oh_data/mice_oh_data.csv") %>% mutate(y = as.factor(y)

# load data directly into tasks for further training
tasks <- list(
  TaskClassif$new("dl_iv", backend = dl_iv_data, target = "y"),
  TaskClassif$new("mf_iv", backend = mf_iv_data, target = "y"),
  TaskClassif$new("mice_iv", backend = mice_iv_data, target = "y"),
  TaskClassif$new("dl_oh", backend = dl_oh_data, target = "y"),
  TaskClassif$new("mf_oh", backend = mf_oh_data, target = "y"),
  TaskClassif$new("mice_oh", backend = mice_oh_data, target = "y")
)

# knn learner
# setting the tunning for parameters, and terminator
knn_learner <- lrn("classif.kknn", predict_type = "prob")
knn_param_set <- ParamSet$new(params = list(ParamInt$new("k", lower = 20, upper = 20)))
terms <- term("none")

# creat autotuner, using the inner sampling and tuning parameter with grid_search
inner_rsmp <- rsmp("cv", folds = 5L)
```

```r
knn_auto <- AutoTuner$new(learner = knn_learner, resampling = inner_rsmp,
                          measures = msr("classif.auc"), tune_ps = knn_param_set,
                          terminator = terms, tuner = tnr("grid_search"))

# use outer sampling (nested sampling)
outer_rsmp <- rsmp("cv", folds = 3L)
design = benchmark_grid(
  tasks = tasks,
  learners = knn_auto,
  resamplings = outer_rsmp
)

# set seed before traing,
# then runs the benchmark, and save the results
set.seed(2020)
knn_bmr <- benchmark(design, store_models = TRUE)


# autoplot auc for all tasks (merged in one plot)
multiplot_roc <- function(models, type="roc"){
  plots <- list()

  # remove x, y axis text, and only keep ticks.
  thm <- theme(axis.text.x = element_blank(), axis.text.y = element_blank())

  # For all tasks we do:
  # extract certain model from benchmarkResult
  # aggregates average AUC value over the model
  # plot the ROC curve with AUC value listed in the title

  model <- models$clone()$filter(task_id = "dl_iv")
  auc <- round(model$aggregate(msr("classif.auc"))[[7]], 4)
  plots[[1]] <- autoplot(model, type = type) + ggtitle(paste("dl_iv:", auc)) + thm

  model <- models$clone()$filter(task_id = "mf_iv")
  auc <- round(model$aggregate(msr("classif.auc"))[[7]], 4)
  plots[[2]] <- autoplot(model, type = type) + ggtitle(paste("mf_iv:", auc)) + thm

  model <- models$clone()$filter(task_id = "mice_iv")
  auc <- round(model$aggregate(msr("classif.auc"))[[7]], 4)
  plots[[3]] <- autoplot(model, type = type) + ggtitle(paste("mice_iv:", auc)) + thm

  model <- models$clone()$filter(task_id = "dl_oh")
  auc <- round(model$aggregate(msr("classif.auc"))[[7]], 4)
  plots[[4]] <- autoplot(model, type = type) + ggtitle(paste("dl_oh:", auc)) + thm

  model <- models$clone()$filter(task_id = "mf_oh")
  auc <- round(model$aggregate(msr("classif.auc"))[[7]], 4)
  plots[[5]] <- autoplot(model, type = type) + ggtitle(paste("mf_oh:", auc)) + thm

  model <- models$clone()$filter(task_id = "mice_oh")
  auc <- round(model$aggregate(msr("classif.auc"))[[7]], 4)
  plots[[6]] <- autoplot(model, type = type) + ggtitle(paste("mice_oh", auc)) + thm
```

```r
  # merge all plots in one plot
  do.call("grid.arrange", plots)
}

multiplot_roc(knn_bmr)
```



From the ROC plots above, we can see that KNN performs with no significant difference between different encoding and missing data handling methods. Moreover, To reduce our computation cost, we decided to take the task with the highest AUC value in the step, being **dl_iv**. In the following sections, we will focus on the task **dl_iv**, and fine-tune the parameters.

KNN parameters

The KNN package included in mlr3 has the following parameters: k (the number of neighbors considered), distance (Parameter of Minkowski distance), kernel (kernel functions used to weight the neighbors). In the following section we will first explore them separately, and then combine the knowledge to perform further fine tuning to improve the model.

K

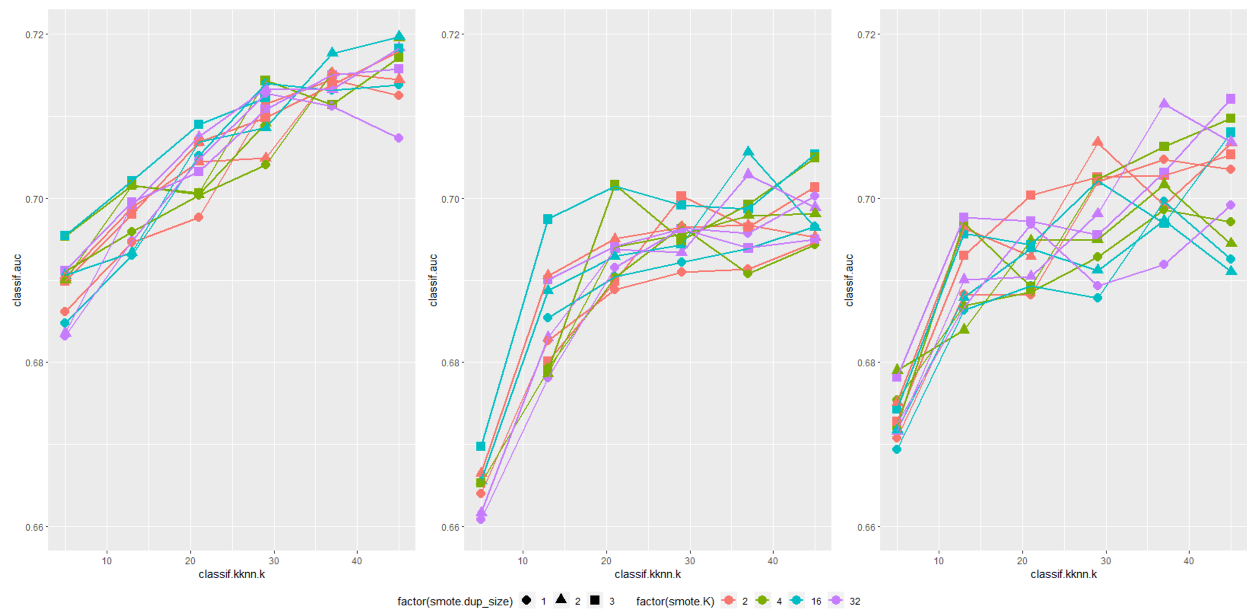
```r
task <- TaskClassif$new("dl_iv", backend = dl_iv_data, target = "y")
```

```r
# knn with 3 different paramSets, to analysis how to do further tuning
knn_lrn <- lrn("classif.kknn", predict_type = "prob")

# only k
param_k <- ParamSet$new(params = list(ParamInt$new("k", lower = 5, upper = 100)))
inner_rsmp <- rsmp("holdout")
```

Using smote to balance data

```r
task <- TaskClassif$new("dl_iv", backend = dl_iv_data, target = "y")

# knn learner
knn_learner <- lrn("classif.kknn", predict_type = "prob")
po_smote = po("smote", dup_size = 6)
lrn_smote <- GraphLearner$new(po_smote %>>% knn_learner, predict_type = "prob")

# setting the tunning for parameters, and terminator
knn_param_set <- ParamSet$new(params = list(ParamInt$new("classif.kknn.k", lower = 5, upper = 45),
                                           ParamInt$new("smote.dup_size", lower = 1, upper = 3),
                                           ParamInt$new("smote.K", lower = 1, upper = 5)
                                           ))


knn_param_set$trafo = function(x, param_set) {
  x$smote.K = round(2^(x$smote.K))
  x
}

terms <- term("none")


# creat autotuner, using the inner sampling and tuning parameter with random search
inner_rsmp <- rsmp("cv",folds = 5L)
knn_auto <- AutoTuner$new(learner = lrn_smote, resampling = inner_rsmp,
                          measures = msr("classif.auc"), tune_ps = knn_param_set,
                          terminator = terms, tuner = tnr("grid_search", resolution = 6))

# set outer_resampling, and creat a design with it
outer_rsmp <- rsmp("cv", folds = 3L)
design = benchmark_grid(
  tasks = task,
  learners = knn_auto,
  resamplings = outer_rsmp
)

# 14:08 -> 14:28, 14:34 ->
# set seed before traing, then run the benchmark
# save the results afterwards
set.seed(2020)
knn_bmr <- benchmark(design, store_models = TRUE)

library(ggplot2)
```

```r
stune_path1 = knn_bmr$data$learner[[1]]$archive("params")
stune_gg1 = ggplot(stune_path1, aes(
  x = classif.kknn.k,
  y = classif.auc, col = factor(smote.K), shape = factor(smote.dup_size))) +
  geom_point(size = 4) + ylim(0.66, 0.72) +
  geom_line(size=1) #+ theme(legend.position = "none")

stune_path2 = knn_bmr$data$learner[[2]]$archive("params")
stune_gg2 = ggplot(stune_path2, aes(
  x = classif.kknn.k,
  y = classif.auc, col = factor(smote.K), shape = factor(smote.dup_size))) +
  geom_point(size = 4) + ylim(0.66, 0.72) +
  geom_line(size=1) #+ theme(legend.position = "none")

stune_path3 = knn_bmr$data$learner[[3]]$archive("params")
stune_gg3 = ggplot(stune_path3, aes(
  x = classif.kknn.k,
  y = classif.auc, col = factor(smote.K), shape = factor(smote.dup_size))) +
  geom_point(size = 4) + ylim(0.66, 0.72) +
  geom_line(size=1)

library(ggpubr)
ggarrange(stune_gg1, stune_gg2, stune_gg3, common.legend = TRUE, legend="bottom", nrow=1)
```

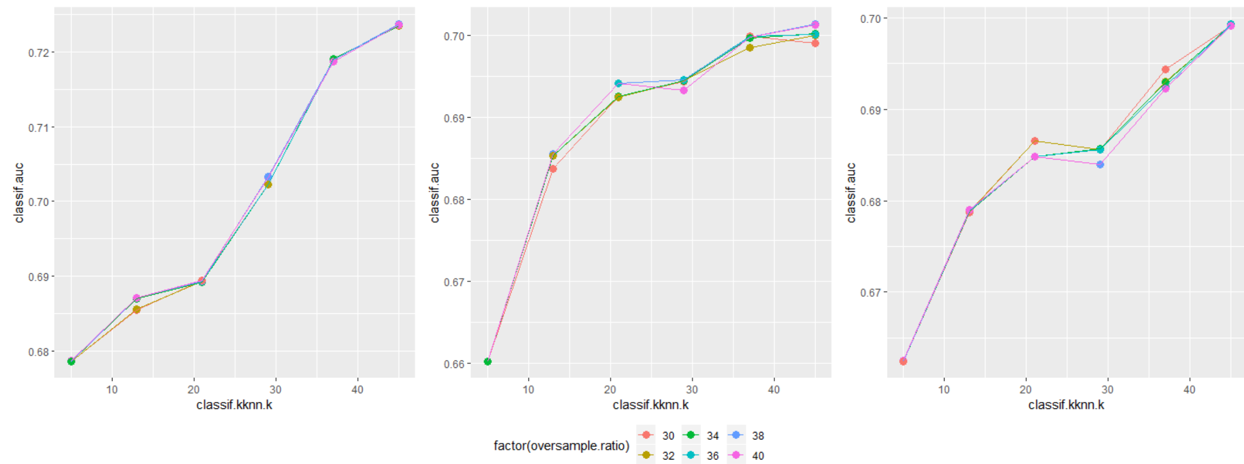


oversampling

```r
knn_learner <- lrn("classif.kknn", predict_type = "prob")
# po_smote = po("smote", dup_size = 6)
po_over = po("classbalancing",
             id = "oversample", adjust = "minor",
             reference = "minor", shuffle = FALSE, ratio = 6)
```

```r
lrn_over <- GraphLearner$new(po_over %>>% knn_learner, predict_type = "prob")

# setting the tunning for parameters, and terminator
knn_param_set <- ParamSet$new(list(ParamInt$new("classif.kknn.k", lower = 5, upper = 45),
                            ParamDbl$new("oversample.ratio", lower = 30, upper = 40)))

terms <- term("none")


# creat autotuner, using the inner sampling and tuning parameter with random search
inner_rsmp <- rsmp("cv",folds = 5L)
knn_auto <- AutoTuner$new(learner = lrn_over, resampling = inner_rsmp,
                          measures = msr("classif.auc"), tune_ps = knn_param_set,
                          terminator = terms, tuner = tnr("grid_search", resolution = 6))

# set outer_resampling, and creat a design with it
outer_rsmp <- rsmp("cv", folds = 3L)
design = benchmark_grid(
  tasks = task,
  learners = knn_auto,
  resamplings = outer_rsmp
)

# 14:08 -> 14:28, 14:34 ->
# set seed before traing, then run the benchmark
# save the results afterwards
set.seed(2020)
knn_bmr <- benchmark(design, store_models = TRUE)

library(ggplot2)

over_path1 = knn_bmr$data$learner[[1]]$archive("params")
over_gg1 = ggplot(over_path1, aes(
  x = classif.kknn.k,
  y = classif.auc, col = factor(oversample.ratio))) +
  geom_point(size = 3) +
  geom_line() #+ theme(legend.position = "none")

over_path2 = knn_bmr$data$learner[[2]]$archive("params")
over_gg2 = ggplot(over_path2, aes(
  x = classif.kknn.k,
  y = classif.auc, col = factor(oversample.ratio))) +
  geom_point(size = 3) +
  geom_line() #+ theme(legend.position = "none")

over_path3 = knn_bmr$data$learner[[3]]$archive("params")
over_gg3 = ggplot(over_path3, aes(
  x = classif.kknn.k,
  y = classif.auc, col = factor(oversample.ratio))) +
  geom_point(size = 3) +
  geom_line() #+ theme(legend.position = "none")

library(ggpubr)
```

```
ggarrange(over_gg1, over_gg2, over_gg3, common.legend = TRUE, legend="bottom", nrow=1)
```



Since we used a binary variable to indicate whether a category is present or not, the max distance can only be 1 or 0. Moreover, other numeric variables have a more significant distance, meaning that they have a more substantial impact on the distance than the categorical data without having a significant correlation with our target variable. To get better results, it would be necessary to either use other ways to handle categorical data better for distance calculation or using other training methods to perform classification instead of KNN.