MC884/MO444 - Aprendizado de Máquina

Experimentação com diversas Técnicas

Erik de Godoy Perillo - RA135582

Universidade Estadual de Campinas

24 de outubro de 2016

1 Introdução

O objetivo do trabalho era experimentar com diversas técnicas de *Machine Learning: K-nearest neighbours, SVM, Neural Networks, Random Forests, GBM.*

1.1 Implementação

A linguagem de implementação escolhida foi o R. Todo o código utilizado no relatório encontra-se na seção 4. Ao longo do documento, linhas do código serão citadas para referência no mesmo. A função `main` (linha 254 da seção 4) executa tudo que é requisitado no enunciado, mostrando os resultados.

2 Metodologia

O pré-processamento pedido é feito na função `pre_proc`, definida na linha 39 da seção 4. As colunas com 30% dos dados faltantes são removidas seguidas das linhas com dados faltantes. Os dados são normalizados com média 0 e desvio padrão 1.

É feita validação *k-fold* externa de 5 *folds* com validações internas de 3 *folds*. Para cada algoritmo, foi feita uma função para achar os melhores parâmetros e uma para a acurácia a cada *fold* externo. Todos os parâmetros requisitados para serem usados nas buscas são definidos a partir da linha 12 da seção 4.

Para todos os algoritmos, é mantida a maior acurácia. No final, todas as maiores acurácias pelo *k-fold* externo são reportadas na linha 380 da função `main`.

- `Knn`: Funções de busca de parâmetros e determinação de acurácia estão nas linhas 81 e 100, respectivamente, da seção 4. O PCA é implementado como pedido, mantendo 80% da variância.
- `SVM`: Funções de busca de parâmetros e determinação de acurácia nas linhas 111 e 126, respectivamente.
- `Neural Network`: Funções de busca de parâmetros e determinação de acurácia nas linhas 148 e 163, respectivamente.
- `Random Forests`: Funções de busca de parâmetros e determinação de acurácia nas linhas 183 e 197, respectivamente.
- `GBM`: Funções de busca de parâmetros e determinação de acurácia nas linhas 214 e 230, respectivamente.

saída relevante do código todo sendo executado pela `main` encontra-se na seção 5.

3 Resultados

Como se observa na saída da seção 5, os algoritmos que obteram melhores acurácias, em ordem decrescente, são: `SVM, Neural Network, GBM, Knn, Random Forest.` Vale notar, entretanto, que todos os algoritmos obtiveram resultados muito parecidos e então não pode-se tirar conclusões definitivas sobre qual o melhor.

4 Código-fonte

```
#packages
library(caret)
library(e1071)
library(class)
library(nnet)
library(randomForest)

#file path of data
data_filepath <- "~/random/secom/secom.data"
labels_filepath <- "~/random/secom/secom_labels.data"

#k for external k-fold
ext_k <- 5
#k for internal k-fold
inn_k <- 3

#knn parameters
knn_ks <- c(1, 5, 11, 15, 21, 25)

#svm parameters
svm_costs <- c(2^-5, 2^0, 2^5, 2^10)
svm_gammas <- c(2^-15, 2^-10, 2^-5, 2^0, 2^5)

#neural net parameters
nn_hidden_layer_sizes <- c(10, 20, 30, 40)

#random forest parameters
rf_nums_features <- c(10, 15, 20, 25)
rf_nums_trees <- c(100, 200, 300, 400, 500)

#gradient boosting machine
gbm_nums_trees <- c(30, 70, 100)
gbm_learning_rates <- c(0.1, 0.05)
gbm_depth <- 5

#wrapper for sprintf
printf <- function(...) cat(sprintf(...))

pre_proc <- function(x, y, empty_ratio_thr=0.3, scale=TRUE)
{
    #filtering out columns with more holes than specified as limit
    empty_num_thr <- floor(empty_ratio_thr*nrow(x))
    col_filter <- apply(x, 2,
        function(col) {sum(is.na(col)) <= empty_num_thr})
    new_x <- x[, col_filter]

    #filtering out rows with holes
    row_filter <- apply(new_x, 1,
        function(row) {!any(is.na(row))})
    new_x <- new_x[row_filter, ]

    new_y <- as.matrix(y[row_filter, 1])

    #scaling x
    if(scale)
    {
        new_x <- scale(new_x, center=TRUE, scale=TRUE)
        #calling pre_proc again to filter out nans since scale produces nans
        ret <- pre_proc(new_x, new_y, empty_ratio_thr, FALSE)
        new_x <- ret$x
        new_y <- ret$y
    }

    return(list("x"=new_x, "y"=new_y))
}

```

```r
#gets minimum number of principal components to keep in order to conserve
#min_var of variance
pca_min_pcs <- function(pcs, min_var)
{
  #getting k minimum number of components required for minimum variance
  pcs_var_cumsum <- cumsum(pcs$sdev^2/sum(pcs$sdev^2))
  min_pcs <- which(pcs_var_cumsum >= min_var)[1]
  #printing result
  #printf("\t-Number of components to keep %.2f%% variance: %d\n",
  # min_var*100, min_pcs)

  return(min_pcs)
}

knn_best_params <- function(x_train, y_train, pca_min_var=0.8, ks=knn_ks,
    cross=inn_k)
{
  #getting principal components maintaining minimum variance percentage
  pcs <- prcomp(x_train, scale=FALSE)
    min_pcs <- pca_min_pcs(pcs, pca_min_var)
  #getting transformation matrix
  transf_mat <- t(pcs$rotation[, 1:min_pcs])

  #transforming data into k dimensions while keeping percentage of variance
  x_train <- as.matrix(x_train) %*% t(transf_mat)

    #performing grid search to find best k
    control <- tune.control(sampling=c("cross"), cross=cross)
    tuning <- tune.knn(x_train, y_train, k=ks, tunecontrol=control)

    return(tuning$best.parameters)
}

knn_accuracy <- function(x_train, y_train, x_test, y_test, k)
{
    #getting prediction
    pred <- knn(x_train, x_test, y_train, k)

    #getting accuracy
    acc <- sum(pred == y_test)/length(y_test)

    return(acc)
}

svm_best_params <- function(x_train, y_train, costs=svm_costs,
    gammas=svm_gammas, cross=inn_k)
{
    #converting to numeric values
    y_train[y_train] = 1
    y_train[!y_train] = 0

    #getting best parameters
    control <- tune.control(sampling=c("cross"), cross=cross)
    tuning <- tune.svm(x_train, y_train, cost=costs, gamma=gammas,
       tunecontrol=control)

    return(tuning$best.parameters)
}

svm_accuracy <- function(x_train, y_train, x_test, y_test, cost, gamma)
{
    #converting to numeric values
    y_train[y_train] = 1
    y_train[!y_train] = 0
    y_test[y_test] = 1
    y_test[!y_test] = 0

    #getting model
    model <- svm(x_train, y_train, scale=FALSE, cost=cost, gamma=gamma)
```

```
136
137     #getting prediction
138     pred <- predict(model, x_test, probability=TRUE)
139     pred[pred >= 0.5] = 1
140     pred[pred < 0.5] = 0
141
142     #getting accuracy
143     acc <- sum(pred == y_test)/length(y_test)
144
145     return(acc)
146 }
147
148 nn_best_params <- function(x_train, y_train,
149     hidden_layer_sizes=nn_hidden_layer_sizes, cross=inn_k)
150 {
151     #converting to numeric values
152     y_train[y_train] = 1
153     y_train[!y_train] = 0
154
155     #getting best parameters
156     control <- tune.control(sampling=c("cross"), cross=cross)
157     tuning <- tune.nnet(x_train, y_train, size=hidden_layer_sizes,
158        tunecontrol=control, MaxNWts=20000)
159
160     return(tuning$best.parameters)
161 }
162
163 nn_accuracy <- function(x_train, y_train, x_test, y_test, size)
164 {
165     #converting to numeric values
166     y_train[y_train] = 1
167     y_train[!y_train] = 0
168     y_test[y_test] = 1
169     y_test[!y_test] = 0
170
171     #getting model
172     model <- nnet(x_train, y_train, size=size, MaxNWts=20000)
173
174     #getting prediction
175     pred <- predict(model, x_test)
176
177     #getting accuracy
178     acc <- sum(pred == y_test)/length(y_test)
179
180     return(acc)
181 }
182
183 rf_best_params <- function(x_train, y_train, mtry=rf_nums_features,
184     ntree=rf_nums_trees, cross=inn_k)
185 {
186     #converting to categorical values
187     y_train <- as.factor(y_train)
188
189     #getting best parameters
190     control <- tune.control(sampling=c("cross"), cross=cross)
191     tuning <- tune.randomForest(x_train, y_train, mtry=mtry, ntree=ntree,
192        tunecontrol=control)
193
194     return(tuning$best.parameters)
195 }
196
197 rf_accuracy <- function(x_train, y_train, x_test, y_test, mtry, ntree)
198 {
199     #converting to categorical values
200     y_train <- as.factor(y_train)
201     y_test <- as.factor(y_test)
202
203     #getting model
204     model <- randomForest(x_train, y=y_train, xtest=x_test, ytest=y_test,
```

```
        mtry=mtry, ntree=ntree)

    #getting accuracy
    conf <- ((model$confusion))
    acc <- (conf[1, 1] + conf[2, 2])/sum(conf)

    return(acc)
}

gbm_best_params <- function(x_train, y_train, depth=gbm_depth,
    n_trees=gbm_nums_trees, shrinkage=gbm_learning_rates, cross=inn_k)
{
    #converting to numeric values
    y_train <- as.factor(y_train)

    #getting best parameters
    fitControl <- trainControl(method="repeatedcv", number=cross, repeats=1)
    gbmGrid <- expand.grid(interaction.depth=depth, n.trees=n_trees,
        shrinkage=shrinkage, n.minobsinnode=c(10))
    tuning <- train(x_train, y_train, method = "gbm", trControl = fitControl,
        verbose = FALSE, tuneGrid = gbmGrid)

    return(tuning$bestTune)
}

gbm_accuracy <- function(x_train, y_train, x_test, y_test,
            n_trees, depth, shrinkage, minobs)
{
    #converting to numeric values
    y_train[y_train] <- 1
    y_train[!y_train] <- 0

    #getting model
    model <- gbm.fit(x_train, y_train, n.trees=n_trees, interaction.depth=depth,
        shrinkage=shrinkage, n.minobsinnode=minobs, distribution="adaboost",
        verbose=FALSE)

    #getting prediction
    pred <- predict(model, x_test, n.trees=n_trees, probability=TRUE)
    pred[pred >= 0.5] = 1
    pred[pred < 0.5] = 0

    #getting accuracy
    acc <- sum(pred == y_test)/length(y_test)

    return(acc)
}

#main method for whole challenge
main <- function()
{
    #reading data
    x <- read.csv(data_filepath, header=FALSE, sep=" ")
    x <- as.matrix(x)
    y <- read.csv(labels_filepath, header=FALSE, sep=" ")
    #transforming y into logical matrix
    y <- y == 1
    y <- as.matrix(y[, 1])

    #pre-processing data
    ret <- pre_proc(x, y)
    x <- ret$x
    y <- ret$y

    knn_best_acc <- 0
    svm_best_acc <- 0
    nn_best_acc <- 0
    rf_best_acc <- 0
    gbm_best_acc <- 0
```

```
274
275     folds <- createFolds(y, k=ext_k)
276     i <- 0
277     for(fold in folds)
278     {
279         printf("on fold n. %d...\n", i)
280
281         #getting train/test folds
282         x_train <- x[-fold, ]
283         y_train <- as.matrix(y[-fold, ])
284         x_test <- x[fold, ]
285         y_test <- as.matrix(y[fold, ])
286
287         #KNN
288         printf("--- on KNN ---\n")
289         printf("\tselecting parameters... ")
290         knn_best <- knn_best_params(x_train, y_train)
291         knn_k <- knn_best$k
292         printf("done. k=%d\n", knn_k)
293         printf("\tgetting accuracy... ")
294         knn_acc <- knn_accuracy(x_train, y_train, x_test, y_test, knn_k)
295         printf("done. accuracy=%.6f", knn_acc)
296         if(knn_acc > knn_best_acc)
297         {
298             printf(" (best so far!)")
299             knn_best_acc <- knn_acc
300         }
301         printf("\n")
302
303         #SVM
304         printf("--- on SVM ---\n")
305         printf("\tselecting parameters... ")
306         svm_best <- svm_best_params(x_train, y_train)
307         svm_cost <- svm_best$cost
308         svm_gamma <- svm_best$gamma
309         printf("done. cost=%.6f, gamma=%.6f\n", svm_cost, svm_gamma)
310         printf("\tgetting accuracy... ")
311         svm_acc <- svm_accuracy(x_train, y_train, x_test, y_test,
312             svm_cost, svm_gamma)
313         printf("done. accuracy=%.6f", svm_acc)
314         if(svm_acc > svm_best_acc)
315         {
316             printf(" (best so far!)")
317             svm_best_acc <- svm_acc
318         }
319         printf("\n")
320
321         #NEURAL NETWORK
322         printf("--- on NEURAL NETWORK ---\n")
323         printf("\tselecting parameters... ")
324         nn_best <- nn_best_params(x_train, y_train)
325         nn_size <- nn_best$size
326         printf("done. size=%d\n", nn_size)
327         printf("\tgetting accuracy... ")
328         nn_acc <- nn_accuracy(x_train, y_train, x_test, y_test, nn_size)
329         printf("done. accuracy=%.6f", nn_acc)
330         if(nn_acc > nn_best_acc)
331         {
332             printf(" (best so far!)")
333             nn_best_acc <- nn_acc
334         }
335         printf("\n")
336
337         #RANDOM FOREST
338         printf("--- on RANDOM FOREST ---\n")
339         printf("\tselecting parameters... ")
340         rf_best <- rf_best_params(x_train, y_train)
341         rf_mtry <- rf_best$mtry
342         rf_ntree <- rf_best$ntree
```

```
        printf("done. mtry=%d, ntree=%d\n", rf_mtry, rf_ntree)
        printf("\tgetting accuracy... ")
        rf_acc <- rf_accuracy(x_train, y_train, x_test, y_test,
            rf_mtry, rf_ntree)
        printf("done. accuracy=%.6f", rf_acc)
        if(rf_acc > rf_best_acc)
        {
            printf(" (best so far!)")
            rf_best_acc <- rf_acc
        }
        printf("\n")

        #GBM
        printf("--- on GBM ---\n")
        printf("\tselecting parameters... ")
        gbm_best <- gbm_best_params(x_train, y_train)
        gbm_n_trees <- gbm_best$n.trees
        gbm_depth <- gbm_best$interaction.depth
        gbm_shrinkage <- gbm_best$shrinkage
        gbm_minobs <- gbm_best$n.minobsinnode
        printf("done.\n")
        print(gbm_best)
        printf("\tgetting accuracy... ")
        gbm_acc <- gbm_accuracy(x_train, y_train, x_test, y_test,
            gbm_n_trees, gbm_depth, gbm_shrinkage, gbm_minobs)
        printf("done. accuracy=%.6f", gbm_acc)
        if(gbm_acc > gbm_best_acc)
        {
            printf(" (best so far!)")
            gbm_best_acc <- gbm_acc
        }
        printf("\n")

        i <- i + 1
        printf("\n")
    }

    printf("FINAL RESULTS:\n")
    printf("\tknn best accuracy: %.6f\n", knn_best_acc)
    printf("\tsvm best accuracy: %.6f\n", svm_best_acc)
    printf("\tnn best accuracy: %.6f\n", nn_best_acc)
    printf("\trf best accuracy: %.6f\n", rf_best_acc)
    printf("\tgbm best accuracy: %.6f\n", gbm_best_acc)
}
```

5 Saída do código

```
FINAL RESULTS:
        knn best accuracy: 0.933333
        svm best accuracy: 0.933674
        nn best accuracy: 0.933674
        rf best accuracy: 0.931035
        gbm best accuracy: 0.933333
```