Exam - Take home

Table of contents

```r
# loading data
financial_data_scaled_exam <- readRDS('data/financial_data_scaled_exam.rds')
```

These exercises are intentionally less directed than the in-class exam to encourage exploration and personal solution/implementation. The goal of the exercises is to explore a method together with corresponding R packages, understand their basic usage then implement it or experiment their behaviour.

1 Exercise: Lasso: choice of lambda using hold out / cross-validation (25% total points)

TO DO at-home

The aim of this exercise is to find an optimal value of the lambda parameter for Lasso regression. You won't use the built-in functions `glmnetUtils::cv.glmnet` or `glmnet::cv.glmnet`, as done for the exam.

- First assume a simpler problem where your data set is split in training and testing set (you can re-use the holdout split you made at the exam).

```r
# YOUR CODE HERE
```

- Fit the Lasso path (more details here) on the training set making use of `glmnetUtils::glmnet` or `glmnet::glmnet` functions

```
# YOUR CODE HERE
```

- using the `glmnet` object and `predict` function, for each lambda of the Lasso path (or better for all lambdas at once) obtain the predicted probabilities on the testing/holdout set

```
# YOUR CODE HERE
```

- using the preceding step you should be able to compute the hold-out error (or criterion if AUC) for each lambda (you can use a `for` loop), then conclude on the best lambda

```
# YOUR CODE HERE
```

- going one step further, replace the hold-out approach by a K-Fold Cross Validation (for each fold of you data, compute the cross-validation error (or criterion) for each lambda (you can use two nested `for` loops)); you can then conclude on the best lambda in terms of mean cross-validation error (or criterion). Finally compare the lambda chosen using your approach with the built-in function `cv.glmnet` described here that was used in the BONUS exercise of the exam. If you want that `cv.glmnet` operates on the same folds/splits as you to allow a closer comparison, you can use the `foldid` parameter of `cv.glmnet` (users can explicitly control the fold that each observation is assigned to via the foldid argument).

```
# YOUR CODE HERE
```

2 Exercise: Decision trees (35% total points)

TO DO at-home

2.1 `CART/rpart`

- Using the data set `financial_data_scaled_exam`, build and plot a decision tree of your choice using `rpart`. You can use default parameters or tweak it and this way learn their effect. (see for example 2. Building the treecin `rpart` vignette here).
- Try to display class probabilities and number of observations per node/leaf (for that explore functions `rpart.plot` or `pdp` from package `rpart.plot` see vignette).

```
# YOUR CODE HERE
```

Explain briefly how the score / probabilities / class are assigned to a single observation using the tree.

- Playing with `rpart` parameters (in particular those inside `rpart.control`), fit, plot, then compare a "larger/deeper" decision tree and a "smaller/shallower" tree in terms of ROC/AUC/prediction on the holdout set from the exam. Looking at the ROC curves, especially those of the smaller tree, you should remark something peculiar with the ROC curve aspect, can you explain?

```
# YOUR CODE HERE
```

- Try to understand the pruning process of a decision tree. You don't have to understand deeply the pruning algorithm but at least what is at stake in the process (see for example 4. Pruning the tree of package `rpart` vignette here or chapter 9. Additive Models, Trees, and Related Methods, p. 308 of Hastie et al. (2009)) and also how to prune a tree with in practice `rpart`. Display the output of the `printcp` function for your "larger/deeper" tree, look at the package documentation to understand the meaning of the columns:

```
# YOUR CODE HERE
```

- Choose an arbitrary terminal number of leaves (closely linked to the number of splits `nsplits`) and the corresponding `cp` parameter, then prune the tree using the `prune` function. and `cp` parameter:

```
# YOUR CODE HERE
```

- Select an optimal `cp` parameter for your tree (you can use a plot given by `plotcp`) and compare to the "large" and "small" models fitted before in terms of ROC curve / AUC.

```
# YOUR CODE HERE
```

2.2 Gradient boosting

Using a gradient boosting package of your choice `gbm` or `xgboost`, assess the hyperparameters (i) number of boosting iterations, (ii) learning rate and (iii) weak learner complexity (decision tree depth, min number of observations) on the financial_data_scaled_exam data.

You can use learning curves (ie misclassification error or AUC on testing set versus number of iterations) or ROC curves.

You can take inspiration from this short example from scikit-learn documentation here which reproduces figures from Hastie et al. (2009) (Chapter 10 Boosting and Additive Trees, figures 10.9/11/12). Time permitting you can also compare the logistic loss (`bernoulli`) with the exponential loss (`adaboost`).

```
# YOUR CODE HERE
```

Example usage of `gbm` (more here, see for example the Figure 3 showing Out-of-sample predictive performance by number of iterations and shrinkage/learning rate):

```
# library(gbm)
# gbm_deg <- gbm(Y~.,
#            data = YOUR_DATA,
#            n.trees = N, # number of boosting iterations
#            distribution = "bernoulli", # loss minimized
#            interaction.depth = 1,
#            /!\ interaction.depth (~#{terminal nodes}+1) <> rpart maxdepth
#                n.minobsinnode = 1, # comparable to rpart minbucket
#            shrinkage = 1, # learning rate
#            bag.fraction = 1) # set at 1 to implement pure boosting
#            # (otherwise stochastic boosting, ie mix of boosting and bagging)
```

Example usage of `xgboost` (see here for the `R` package):

```
#library(xgboost)
# xgb_deg <- xgboost(data = as.matrix(YOUR_PREDICTORS),
#            label = as.vector(YOUR_TARGET),
#            max.depth = 1, # comparable to rpart maxdepth
#            eta = 1, # learning rate
#            nthread = 1,
#            nrounds = num_tree, # number of boosting iterations
#             min_child_weight = 0, # similar but not comparable to rpart minbucket
#             objective = "binary:logistic", # loss minimized
#             lambda = 0, # set at 0 to avoid L2 penalization
#             tree_method = "exact")
```

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning*. Springer New York. https://doi.org/10.1007/978-0-387-84858-7