UWEO StatR201 – Winter 2013: Homework 6

Due: Friday March 22, 11:59 PM (grace period up to 2 days, with notification)

Assaf Oron, assaf@uw.edu

Reading related to this assignment: Lectures 9-10; Hastie-Tibshirani-Friedman, Sections 4.5, 12.1-12.3, 11.3-11.7.

- Please submit online in the class dropbox. Please submit either ***.pdf (with code pasted verbatim or separately attached), or *.rmd.** (tip: to save time when using knitr, use `'cache=TRUE'` in the chunk header for any code chunks that run big simulations – this will prevent them from re-running each time you recompile the code).
- **Starred (*) questions and question-parts are not required.** You may submit them if you choose, or do any part of them without submitting.
- **Grading is determined chiefly by effort, not by correctness. If your submission shows evidence of independent, honest effort commensurate with the amount of homework assigned – you will receive full credit.**

1. wine Dataset with Mid-range Classifiers and proper Validation

For this question, choose one of the "mid-range" classifiers shown on Lecture 9 (random forests, boosting or '`kknn`').

a. Tune the classifier on the `wine` training set, using at least 2 tuning parameters simultaneously. You are allowed to use either k-fold CV (not leave-one-out), or bootstrap with *out-of-bag* predictions. **Make sure to use proper validation: no "diffusion" of information from the left-out data to the in-sample data.** For example, do not scale the features over the entire dataset before sending them through the tuning; rather, either write out a validation in which featured are scaled using the in-sample data parameters only (you can take advantage of `normTrainingAndTestSet` in the `KSNNS` package, similar functions in `caret`, or write your own) – or not scale the data at all, if the classifier you use auto-scales the data on its own.

If the latter is true, you may also take advantage of the `tune` function (from `e1071`) or the `train` function (from `caret`) for the entire tuning exercise.

b. Predict the test data once it is uploaded, using your tuned method. Print the confusion matrix as well as the overall test error rate. Before predicting, present some descriptives and determine whether the test dataset is qualitatively similar to or different from the training dataset.

2. Dataset Manipulation and Pre-Processing.

Download the 'smarTrain.csv' dataset of smartphone-recorded movement data.

a. Lecture 10 shows how to use `ddply` (from package `plyr`) to scale features by subject mean/SD. Use `ddply` to produce a dataset, where each individual row represent the feature-level averages for a specific subject and class. That is: the first row might have the averages of Subject 1 Class 1 data, the second row Subject 1 class 2, etc. The output dataset will have the same number of columns as the raw data, but far fewer rows. **Explore the summarized data frame a bit, showing at least one plot.**

Hint: look up the `ddply` utility `summarise`. **Note: this is not an empty exercise; such a descriptive data frame will be very useful in understanding the data patterns and adapting the right prediction tool for them.**

b*. Lecture 10 shows feature filtering "by hand" based on features' ability to discriminate among the 3 walking classes or among the 3 stationary classes. What was **not** done, is inspect and thin-out **highly correlated** features. Use `findCorrelation` (from `caret`) to identify these features (after the between/within filtering), first using the default cutoff of 0.90 correlation, then changing it to 0.95. Read the function's help to make sure you understand what it does. How many features are removed in each case? **Note: the function does not eliminate the features, just returns their indices.**

<u>3. Smartphone Dataset classification using Advanced Methods</u>

a. Divide the training set, after scaling-by-subject, **randomly** into 7 groups of 3 subjects each, and carry out 7-fold CV, via `kknn` and **one** of the advanced methods from Lecture 10 (SVM, or neural nets using RSNNS). Tune at least two parameters for each method; you don't have to do it simultaneously (tune the "coarse" paramter first, then the "fine" one). The parameters don't have to be "off-the-shelf"; for example, you might consider the number of neural-net layers, or feature weighting.

Make sure to use proper cross-validation. Here we don't have to worry about scaling issues (since data are already scaled by subject). But we <u>do</u> have to carry out feature filtering separately for each CV group. You need to use at least some feature-filtering, such as the type shown in class (by ability to discriminate). If you wish, you may use additional or alternative filtering (by correlation, etc.; see 2b*).

Note: given the subject-based grouping, you will likely need to write the CV routines "by hand" rather than use a ready-made tuning utility. Note the peculiar syntax of preparing a training and validation set in RSNNS, as shown in class.

Score the CV results for both methods as usual (including the confusion matrix), and choose the "best" two models for test.

b. When the test dataset is uploaded, explore it a bit (see 2a above). Is it qualitatively different in any conspicuous way? Then predict and score your tuned methods.

c*. Repeat the exercise in (a), *for the entire un-filtered feature set.* Compare the runtime (via `system.time`) and the results with the filtered version. Attempt to improve runtime for both options using a parallel processing package.