Econ 5253 - Spring 2024 **Problem Set 10** **Due: Apr. 16 beginning of class**

This problem set will give you practice in using cross-validation to tune classification models using four of the Five Tribes of Machine Learning: trees, neural networks, naive Bayes, and k-nearest neighbor / support vector machines.

As with the previous problem sets, you will submit this problem set by pushing the document to *your* (private) fork of the class repository. You will put this and all other problem sets in the path `/DScourseS24/ProblemSets/PS10/` and name the file `PS10_LastName.*`. Your OSCER home directory and GitHub repository should be perfectly in sync, such that I should be able to find these materials by looking in either place. Your directory should contain at least three files:

- `PS10_LastName.R` (you can also do this in Python or Julia if you prefer, but I think it will be much more difficult to use either of those alternatives for this problem set)
- `PS10_LastName.tex`
- `PS10_LastName.pdf`

1. Type `git pull origin master` from your OSCER `DScourseS24` folder to make sure your OSCER folder is synchronized with your GitHub repository.
2. Synchronize your fork with the class repository by doing a `git fetch upstream` and then merging the resulting branch. (`git merge upstream/master -m "commit message"`)
3. Install the following machine learning packages if you haven't already:
   - `rpart`
   - `e1071`
   - `kknn`
   - `nnet`
   - `kernlab`
4. Start your code file by importing the starter code I have provided you at https://github.com/tyleransom/DScourseS24/blob/master/ProblemSets/PS10/PS10starter.R. This starter code reads in the adult income data from the UC Irvine datasets repository. The goal of this problem set is to compare the 5 Tribes in terms of their ability to classify whether someone is high-income or not. Thus, the target variable for this exercise will be `income$high.earner`.
5. Using the `tidymodels` library (or a similar resource in Python or Julia), train, cross-validate and compute the accuracy of the predicted "high earner" variable in the test

set. We will dispatch with `recipes` since we won't do any feature engineering in this problem set.

Assume the following architecture in `parsnip`:

- classification mode
- **3-fold** cross-validation
- various "specifications" and "engines" (algorithms):
  1. Logistic regression: `logistic_reg()` with engine `"glmnet"`
  2. Trees: `decison_tree()` with engine `"rpart"`
  3. Neural network: `mlp()` with engine `"nnet"`
  4. kNN: `nearest_neighbor()` with engine `"kknn"`
  5. SVM: `svm_rbf()` with engine `"kernlab"`

6. Each algorithm has hyperparameters that will need to be cross validated:

- Logit model
  - `penalty` (this is the λ of the LASSO model just like in PS9)
- Tree model
  - `min_n`, which is an integer ranging from 10 to 50 (governs minimum sample size for making a split)
  - `tree_depth`, which is an integer ranging from 5 to 20 (governs maximum tree depth)
  - `cost_complexity`, which is a real number ranging from 0.001 to 0.2 (governs complexity of the tree)
- Neural network model
  - `hidden_units`, which is an integer ranging from 1 to 10 (governs number of units in hidden layer)
  - `penalty`, which acts like λ in the LASSO model
- kNN
  - `neighbors`, which is an integer ranging from 1 to 30 (governs the number of "neighbors" to consider)
- SVM
  - `cost`, which is a real number in the set $\left\{2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{10}\right\}$ (governs how soft the margin of classification is)

- `rbf_sigma`, which is also a real number in the set $\{2^{-2}, 2^{-1}, 2^0, 2^1, 2^2, 2^{10}\}$ (governs the shape [variance] of the Gaussian kernel)

For hyperparameters that are real-valued (i.e. can take on any real number), we can define a grid like we did with LASSO:

```
lambda_grid <- grid_regular(penalty(), levels = 50)
```

but when the hyperparameters are discrete (i.e. can only take on integer values), we should instead create a data frame with the values that we want it to take on. For example, with kNN, we would set

```
knn_grid <- tibble(neighbors = seq(1,30)
```

7. Now tune the models. Use the accuracy as the criterion for tuning.

8. Once tuned, apply the optimal tuning parameters to each of the algorithms. Then train the models, generate predictions, and assess performance. (Just like in PS9).

9. As a **table** in your .tex file, report the optimal values of the tuning parameters for each of the algorithms. How does each algorithm's out-of-sample performance compare with each of the other algorithms?

10. Compile your .tex file, download the PDF and .tex file, and transfer it to your cloned repository on OSCER. There are many ways to do this; you may ask an AI chatbot or simply drag-and-drop using VS Code. Do **not** put these files in your fork on your personal laptop; otherwise git will detect a merge conflict and that will be a painful process to resolve.

11. You should turn in the following files: .tex, .pdf, and any additional scripts (e.g. .R, .py, or .jl) required to reproduce your work. Make sure that these files each have the correct naming convention (see top of this problem set for directions) and are located in the correct directory (i.e. `~/DScourseS24/ProblemSets/PS10`).

12. Synchronize your local git repository (in your OSCER home directory) with your GitHub fork by using the commands in Problem Set 2 (i.e. `git add`, `git commit -m "message"`, and `git push origin master`). More simply, you may also just go to your fork on GitHub and click the button that says "Fetch upstream." Then make sure to pull any changes to your local copy of the fork. Once you have done this, issue a `git pull` from the location of your other local git repository (e.g. on your personal computer). Verify that the PS10 files appear in the appropriate place in your other local repository.