Machine Learning: Final Project

Modeling in Subgroups: One Model or Many?

Large data sets can include a variety of subgroups. The qualities of these subgroups may be quite different from each other. Each subgroup may have a different relationship between the outcome and the predictors in a supervised learning model. This raises a question of how to generate the best predictions. Some options (among others) would include:

- *One Model*: All of the subgroups are aggregated into a single data set that is used to inform the construction of a single model.
- *Many Models*: A separate model is constructed for each subgroup based upon the corresponding subset of the overall data.

These differing approaches have their respective advantages and disadvantages:

- *One Model*: It is easier to build, use, revise, and maintain one model. A larger overall sample size can be utilized. However, a single model will assume that each prediction's association with the outcome is similar across all of the subgroups.
- *Many Models*: Modeling within a subgroup can provide a customized estimate of the relationships between the predictors and the outcome in that subgroup. This allows for differing effects across the models. However, each subgroup's model will necessarily be based on a smaller sample size. Developing, using, and revising multiple models can also require substantially greater labor.

Other modeling strategies can provide a strategy in between these two approaches. Utilizing interaction terms between the subgroup and the other variables can allow for the development of one model while still producing differing relationships in the subgroups. However, such a model can very quickly create numerous terms that can be difficult to utilize, and the ratio of the sample size relative to the number of predicting factors will reduce as more interactions are introduced.

In this project, we will take a practical approach to evaluating the question of which modeling strategy leads to the most effective predictions. Using a real data set, we will design an experiment to answer the question of whether to use one model or many models.

The Setting: Predicting Patient Survival

For this project, we will be using patient health data from MIT's GOSSIS (Global Open Source Severity of Illness Score). This dataset was used in the 2020 Global Women in Data Science hackathon hosted by Kaggle (https://www.kaggle.com/c/widsdatathon2020/overview).The dataset includes information about more than 130,000 Intensive Care Unit (ICU) visits. Our goal is to build a model to predict patient survival given these ICU visit information.

The Data

You can access the data through the competition site here (https://www.kaggle.com/c/widsdatathon2020/data). The file labeled "training_v2.csv" is our training data and "unlabeled.csv" is the competition evaluation dataset (with no labels). Refer to "WiDS Datathon 2020 Dictionary.csv" for detailed description of the features.

Training and Testing Sets

First, we randomly split our data ("training_v2.csv") into training & testing sets (80/20). For the One Model approach, you train & tune your model using the training set and predict on the testing set. For the Many Models approach, you'll split both your training set and testing set by subgroups, train & tune your model using the subgroup training set only and predict on your subgroup testing set.

Adversarial Validation

In some instances, our training & testing sets do not appear to come from the same distribution (not i.i.d.). For example, some features may have high cardinality, and when we randomly split them into two sets, the features do not appear to come from the same distribution. We should omit those features from our modeling as they would lead to biased predictions. An approach to identify these features is called *Adversarial Validation*, you can read more about it here (https://www.kdnuggets.com/2016/10/adversarial-validation-explained.html). Use adversarial validation to identify and remove features that may not generalize well to the overall dataset.

The Experiment

Each project will independently assess the question of whether to use one model or many models by fitting a variety of machine learning procedures.

Outcome

The outcome of each model will be a binary indication of whether a patient survived (1 or TRUE) or not (0 or FALSE).

Classification Approach

Each model will be used to classify whether a patient survived (hospital_death). The values returned will either be:

- 1 (or TRUE): The patient survived.
- 0 (or FALSE): The patient did not survived.

Your selected models must either directly perform this classification, or you must recode the predicted probabilities into classifications.

Categories of Models

You must choose models from 5 different categories out of the following list, with recommended packages in R:

- K Nearest Neighbors (Package: class; Function: knn)
- Logistic Regression (Function: glm)
- Lasso, Ridge, or Elastic Net Regression (Package: glmnet; Function: glmnet)
- Decision Tree (Package: rpart; Function: rpart)
- Random Forest (Package: randomForest; Function: randomForest)
- Boosting Models (Package: gbm; Function: gbm or Package: xgboost; Function: xgboost)
- Support Vector Machines (Package: e1071; Function: svm)
- Neural Networks (Package: nnet; Function: nnet)

You are not required to use the recommended packages; other packages are acceptable.

Experimental Subgroups: Age

The age variable will be used as the basis of assessing the relative merits of fitting one model (on all of the data) or many models (one for each subgroup of age).

The age subgroups can be defined as: [0-50), [50-60), [60-70), [70-80), [80-100). The table below summarizes the number of observations by age groups. Note that there are 4228 missing values in age variable, please omit those observations from the model.

Age Group	# Obs.
[0-50)	18012
[50-60)	15918
[60-70)	20052
[70-80)	19463
[80-100)	14040
NA	4228

Each of the 5 models you choose will be fit in two ways: as One Model and as Many Models.

- *One Model*: You will fit a single model that includes all of the training data, including all of the subgroups of age.
- *Many Models*: You will fit separate models on each subset of the training data, one for each of the subgroups of age.

These two models should be as comparable as possible. The list of predictors and formula should be the same. The only exception is that age cannot be included as a predictor within any subgroup of the Many Models approach. Furthermore, while interaction terms are allowed, no interactions between age and other variables may be used for the Many Models approach. Interactions are allowed for the One Model approach. This will ensure that the comparison between One Model and Many Models is as fair as possible. Furthermore, each corresponding fit of One Model and Many Models must also have the same specification of the parameters (or use the same automated approach to select the parameters).

Building All of the Models

With 5 categories of models selected and 2 forms (One Model and Many Models), you will generate 10 different sets of predictions. This will include 5 sets of predictions with the One Model approach and 5 corresponding sets of predictions from the Many Models approach.

Evaluating the Models

For any single version of any modeling category, the following steps will be taken:

- *Fit to Training Set*: The model will be fit to the relevant training data. One Model approaches will be fit to all of the training data, while Many Model approaches will have separate fits for each subgroup of age in the training data.
- *Predicted Classifications*: Each model will generate predictions on the relevant testing data. One Model approaches will make predictions on all of the testing data, while Many Model approaches will generate separate predictions for each subgroup of testing data (using the corresponding model fit on this subgroup of age in the training data). Predictions from Many Models will be aggregated to form one set of overall predictions for the entire testing set.
- *Evaluation*: Each set of predictions will be compared to the outcomes of the testing set. The proportion of correct classifications (a decimal number between 0 and 1) will be used as a quality metric. Larger values of this proportion indicate better predictions.

A Scoreboard

You are required to display the results of your experiments in a scoreboard of the following form:

The Modeling Type should include values that correspond to the algorithms you select in the 5 categories.

This table should be sorted in decreasing order of one of the proportion columns to display the best overall prediction in the first row. Please round the results to a reasonable number of digits (e.g. 2, 3, or 4). Depicting the table using the datatable function in the DT package will create a sortable HTML table.

Details

- *Individual Project*: This project will be completed individually.
- *Collaboration Policy*: You may not discuss your project with others. It is OK to contact the teaching staff for reasonable clarifications, but the work you perform must be your own.
- *Sources*: You may use any publicly available written resources to help, but please cite your sources.
- *Materials to Submit*: You must write a report in RMarkdown format. Please provide the code file (in Rmd format) and the output file (HTML preferred). We are supplying a template file to help you get started. The final report should be of moderate length (roughly 2000-3000 words, not strictly enforced). Include all of the code you used, and display any relevant tables, plots, or other supplementary materials.

In your report, you need to give a detailed explanation of each step you take to arrive at your solution. Please give a justification or explanation of the results you obtain. The reports should also include the lessons you may have learned from doing the project.

The Report

Each of the 5 categories of models you construct should be summarized in a separate section. The overview should include a few paragraphs of written content along with the programming code that was used. Show the code that you use to fit the model to the training data, generate predictions on the testing set, and evaluate the accuracy of the results. Provide any details about the selected parameters. Explain why you

selected this technique, a few details about how it works, and a short discussion of its advantages and disadvantages.

The report should be written in a style that emphasizes the clarity of the explanation. While we encourage you to show your code in the report, the written content should be approachable and simple to understand. It should include an Introduction section and conclude with a Discussion section. Briefly explain the results of the models and what conclusions you can draw from the findings. You might also explore some related questions: How different were the results for the One Model and Many Model approaches? How much did the sample size of the subgroups impact the results? Are you concerned about the possibility of fitting too many models? Or feel free to propose and address some other questions.

Grading

The report will be graded based on a variety of factors:

- *Data Splitting & Model Evaluation*: Training and evaluating performance for different model types.
- *The Quality of the Implementation*: Generating code that builds conceptually sound models without errors.
- *Coding Practices*: Writing code with a good style.
- *Reproducibility*: Generating results in a manner that can be easily understood and reproduced by others.
- *Playing by the Rules*: Creating solutions that fit the specified requirements.
- *Quality of Communication*: How well the methods and results are explained and presented.

We will not specifically grade the reports based on the overall accuracy of the methods. However, some reasonable standard of quality should be met.

It is more important to obtain high-quality results within the rules we specified and to provide clear descriptions of your work. However, we will not assign additional credit for exhaustive exploration, extra pages, or a larger number of models. You may feel free to consider these possibilities in your own time, but the report should focus only on the 5 selected models and the 3 selected versions you choose.

Computational Hints

The sample size and number of features in the data can create computational challenges. It is not unreasonable for some models to run for a few minutes or more. Some techniques require greater computational resources than others. For these circumstances, it can be a good practice to gradually build up your models. Start with a relatively small sample size (e.g. 100-500 randomly selected rows). Test out the methodology to make sure that the smaller model is working properly. You can use the Sys.time() function to record the time before and after a model is run. See how much time elapses at different sample sizes. Try to get an idea of how long the model will take to run when you provide the intended sample size. It is also fair to select models that are easier to implement if the computations prove to be burdensome.

We recommend that you organize your data with meaningful names that will help streamline your efforts.

One tricky aspect of generating predictions is understanding the workings of Random Forest's predict function. Each modeling technique has its own associated prediction method. For instance, a Random Forest model from the randomForest package has a prediction method called predict.randomForest. It is worthwhile to study the help files for these methods, as sometimes the parameters to supply have some variations in their names or forms. Then, it is also worthwhile to examine the output of the predictions to ensure that they match your expectations. Usually supplying the parameter *type* in the prediction will lead to the

correct (numeric) output, although this may need further steps to ensure that the result is a classification. Other types could lead to output that is not of the correct form to evaluate the method's predictions.

Because you will be fitting the same model on many subgroups, we recommend that you develop functions that will simplify your work. This will also improve the reproducibility and readability of your code. This could include the following steps:

- For each modeling category, write a customized function that will fit that model on a specific data set and generate predictions on the testing set.
- Then, for any single one of the models, it would help to write a function that fits that model to each of the subgroups. This could be as simple as using a for loop to iterate through subgroups and calling the customized function for that model described in the previous point.

With all of this in mind, it should be possible to have all of the work proceed in a relatively small number of processing functions:

- *Modeling Functions*: This could up to 1 per model for a total of 5.
- *Iteration Function*: This would be 1 function that runs through the subgroups.
- *Scoring Function*: This could be 1 function to compute the percentage of correct classifications.

Meanwhile, it should be emphasized that generating 20 sets of predictions may be time consuming. Random Forest models in particular may process quite slowly, especially when a large number of predictors are used. Re-running a report in RMarkdown could require substantial time. Be mindful of this, especially as you near the deadline. It may help to save separate files of your versions so that your earlier work is not overwritten.

We also recommend creating a report that is easy to read and understand. Any lengthy technical outputs in the modeling process should not be included. Instead, all of the technical aspects of the report should be used to help convey the story of your work. Suppressing some elements of your code's output can be achieved either by setting values like echo = FALSE, message = FALSE, warning = FALSE, etc. in the code chunks. Alternatively, the capture.output function can be used to remove printed coding messages (e.g.Â from the train function in the caret library).