15.071C: The Advanced Analytics Edge

Homework Assignment #4

Due on Wednesday, April 5, 2017

1 Problem 1: MITx (36 points)

Complete the problem "Separating Spam from Ham" in Assignment 4 on MITx. All other problems in Assignment 5 on MITx are optional and will not count towards your grade.

2 Problem 2: Developing a framework for regression in Julia (36 points)

Use the instructions that follow to write and test a formulation in Julia for a framework for linear regression. Include a (commented) copy of your code, along with your results in a PDF file you upload to Stellar.

Points are marked in parentheses before subproblems.

And remember - google is your friend :-)

2.1 Preliminary matters (6 points)

1. (1) Install Gurobi. You should all be able to get a free academic license here:

   (`http://www.gurobi.com/registration/academic-license-reg`) and download the software (`http://www.gurobi.com/registration/download-reg`). Then add the Gurobi package to Julia (`Pkg.add("Gurobi")`) and use it like we have with Cbc (replacing `CbcSolver` with `GurobiSolver`). If you really are unable to get Gurobi running, unfortunately, you won't be able to model the objective the way we want to. We will give you an alternative objective when we get to that step. But Gurobi

is really useful (as you can already tell, it can handle more types of problems!) so give installing it a show.

2. (1) Load packages `JuMP` and `Gurobi` (or `Cbc` if you need to) with the `using` command.
3. (1) Set directory to folder that contains data files with `cd()`.
4. (1) Read in train, validation, and test files using `readcsv()`.
5. (2) Separate X and Y variables from each set (to form `x_train`, `y_train`, `x_test`, `y_test` etc.). The **last** column of each dataset is the dependent variable.

2.2 Data cleaning (7 points)

1. Normalize each column in the training data.
   (a) (2) Compute the mean of `y_train`, and each column of `x_train`. (Hint: there are a few ways to apply operations "column-wise". You could loop through column indices. You could use a second argument in many functions that specifies along which dimension to apply the function (look at ?mean for an example).)
   (b) (1) Subtract these mean values from `y_train` and each column. (Now the mean of each column should be 0.) (Hint: again you have some options - you could loop over the columns, or look into the dot operator (look at ?./ for an example).)
   (c) (1) Compute the norm of each column of `x_train` (you don't need to do this for `y_train` this time).
   (d) (1) Divide each column of `x_train` by its norm. (Now the norm of each column of `x_train` should be 1.)
2. (2) Use the column means and norms you just calculated in on the training set to normalize the validation and test set as well. (That is, subtract `y_train_mean` from `y_validation` and `y_test`, subtract `x_train_mean` from `x_validation` and `x_test`, and divide `x_validation` and `x_test` by `x_train_norm`.)

2.3 Starting the model (4 points)

1. (1) Define the constant `D` to be the number of independent variables. (Hint: use `size()` on `x_train`.)
2. (1) Create a model objecting using `Model()` and the solver you intend to use (probably Cbc).
3. (1) Define your `Beta` variables using `@variable`, to have length `D`.

4. (1) Define binary indicator variables `z` of the same length to capture whether each variable in the model will be used.

2.4 Adding constraints (7 points)

1. What constraints relate `Beta` and `z`? You should review the lecture notes for Lecture 9 if you need help.
   - (1) Write a comment in your code explaining the purpose of the `z` variable.
   - (1) What does the "big M" in the formulation represent? Explain in a comment how it works. (Hint: google "big M method" for more info)
   - (2) Fix a value for your "big M" and formulate the constraints you need using `@constraint`.
2. (1) Write a constraint that enforces sparsity in your model. Set the maximum number of variables used to 5.
3. (1) Assume that the following pairs of variables are highly correlated: (1 and 2), (5 and 6). Write constraints to prevent multicollinearity in the model. (Again, see lecture notes for help with this.)
4. (1) Write a constraint to exclude the solution where the first five variables are all used.

2.5 Defining the objective (4 points)

1. (2) Write out in the comments an expression for the sum of squared error we want to minimize, using `y_train`, `x_train`, and the variable `Beta`.
2. (2)

   If you have Gurobi: Formulate the expression in Julia and add it using `@objective`. (Hint: you'll want to use two layers of `sum()` commands, one that sums over the rows of `y_train`, and one that sums over the columns of `x_train`. Take your time working this out - it can be tricky.)

   If you are using Cbc: Replace the squared term on the expression with an absolute value (so you'll get a sum of absolute error terms). To program this, you will need to add a new variable to represent these error terms, and add constraints so they are larger than the value inside the absolute value bars, and larger than negative one times that term (this will look a lot like what you did with Beta and z). Then the objective will be to minimize the sum of these absolute error variables.

2.6 Running the model (8 points)

1. (1) Solve the model using the `solve()` command, and print out the status.
2. (1) Use `getvalue` to save the value of `Beta` to a new array.
3. (2) Create predictions for `y_test` using `x_test` and your saved `Beta`.
4. (2) Calculate the sum of squared error of your model on the test set.
5. (1) Compute the sum of squared error of the baseline model on the test set (be careful here about what the baseline model is!)
6. (1) Compute the R2 of your model.

2.7 EXTRA CREDIT (5 points for each)

These are challenging and open-ended problems. Select one that you think might be interesting, especially one that might be helpful for your project. Significant credit will be given to thoughtful approaches to the problems (so comment your code!!!) even if there are technical mistakes or you aren't able to run the final product.

1. (5) Change the response variable to a binary one (say by separating into values that are higher or lower than the mean value.) Change the model to work for logistic regression. Run the model, and report your error on the test set. (Note: you may need to use a different solver like KNITRO to handle the non-linear objective function.)
2. (5) Add additional variables that are non-linear transforms of the old variables (e.g. squares, logs, square roots) and show how to include them in the model, and prevent the model from selecting multiple representations of each variable. Report your error on the test set.
3. (5) Show how you might select the sparsity (maximum number of variables used) programmatically. Run a loop over various sparsity levels, and select the sparsity that minimizes out-of-sample error on the **validation** set. Then report your error on the test set.