# Readme for modeling 7Q10s using machine learning

https://doi.org/10.5066/F7CR5S4T

*scworland@usgs.gov*

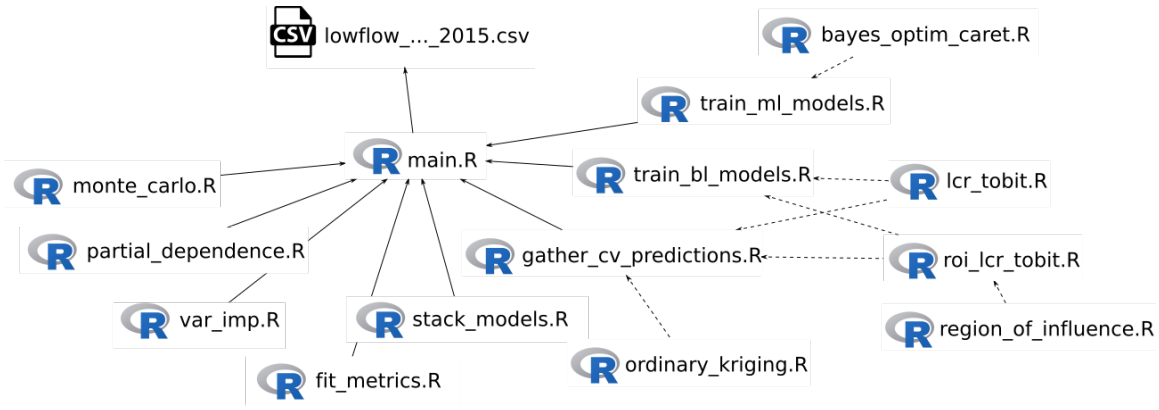
*7/5/2017*

## Contents



### Introduction

This readme file is intended to serve as a navigation guide for the model archive associated with the paper (citation information). The basic file contents and dependencies look like:




The arrows should be interpreted as "output dependency flow". The steps should be run in roughly clockwise order. Each primary step can be run directly with "main.R" file via custom R functions that have the same name as the othe R files in the figure above. Solid lines indicate primary functions that are sourced directly within "main.R" file, and the dashed lines indicate secondary functions that are sourced by the primary functions. For example, the "train_ml_models.R" file is sourced and run directly by "main.R", but "train_ml_models.R" is depedent on "bayes_optim_caret.R". The idea behind structuring the model archive in this way is that output dependencies can be traced directly back to the input data csv file titled "lowflow_sc_ga_al_gagesII_2015.csv". The steps from the data file to the output should be completely reproducible[1] if functions are run in sequential order.

[1]With the exception of possible stochastic differences in model training. This is discussed in more detail in lower sections

## Quick start

Although all the scripts are included in the model archive, most of them need not be run to recreate a bulk of the important results. The quickest way to get started is by opening the "main.R" file and setting up the work environment. This includes installing pacakges and setting the working directory.

```
# Install all packages required for entire analysis. The libraries needed for
# each function are loaded in the function script.
install.packages(c("dplyr","reshape2","randomForest","kknn","caret",
                   "xgboost","kernlab","Cubist","glmnet","devtools","AER",
                   "leaps","doMC","ICEbox","geoR"))

# install PUBAD package off of github
devtools::install_github("wfarmer-usgs/PUBAD")

# load libraries needed for main.R script
library(dplyr); library(PUBAD)

# set working directory
setwd("~/set/working/directoy")
```

The next step is to load the csv file and prepare the data as input for the functions. The only packages that are required for this step are `dplyr` and `PUBAD`.

```
# Load csv file: note, data should be in a "data" folder in working directory
data_full <- read.csv("data/lowflow_sc_ga_al_gagesII_2015.csv",
                      header=T,na.strings = "-999") %>%
  setNames(tolower(names(.))) # make column names lower case

# Transform 7Q10 response variable
area <- data_full$drain_sqkm
ln7q10_da <- log((data_full$y7q10+0.001)/area)

# use functions from PUBAD to cull covariates
expVars <- data.frame(gages = data_full$staid) %>%
  getBasinChar()

# create model data using clean basin chars
model_data <- expVars$cleanBCs %>% # start with basin chars
  mutate(CLASS = as.integer(as.factor(CLASS))) %>% # change class to integer
  mutate_each(funs(as.numeric)) %>% # make everything numeric
  scale() %>% # convert to z-score: (x-mu)/sigma
  as.data.frame() %>% # make data frame
  setNames(tolower(names(.))) %>% # make sure colname are lower case
  mutate(y=ln7q10_da) %>% # add the transformed response variable from above
  select(y, class,lat_gage:aspect_eastness) # reorder column positions
```

The raw 7Q10s are divided by the drainage area of the basin and log transformed prior to modelling. Because some of the 7Q10's are zero, a small, arbitrary constant of 0.001 was added to each 7Q10 value prior to taking the log transform. This data transformation for the baseline models is slightly different, and is explained in the "Description of train_bl_models" section below. After the data is loaded, the first steps is to find the optimal hyperparameters of for all of the models using random search and bayesian optimization. This process will take over 24 hours for all of the models, and we already provide the optimal values in the "gather_cv_predictions.R" file. You can go ahead and jump straight to this step. The `gather_cv_predictions()` function calculates

prediction for each basin and model in a leave-one-out cross validation framework. With that being said, this function will still take ~45 minutes to run.

```
# gather cross validation predictions
source("scripts/gather_cv_predictions.R")
cv_preds <- gather_cv_predictions(model_data,data_full)
```

The next step is to build the ensemble-stacked model using the cross validated predictions calculated in the step above. The output, `all_preds`, is a matrix of the cross validation predictions for all the models and the actual observations. This is the result that is used for the bulk of the analysis.

```
# stacked regression
source("scripts/stack_models.R")
all_preds <- stack_models(cv_preds)
```

Calculate fit metrics using the the cross validation predictions,

```
# calculate fit metrics
source("scripts/fit_metrics.R")
model_error <- fit_metrics(all_preds,data_full)
```

The final steps for this truncated version of the analysis is to calcualte variable importance, partial dependence, and make some plots. The paper includes a monte carlo analysis. I do not suggest you try to run the full monte carlo[2], but the script is included for inspection.

```
# calculate variable importance
source("scripts/var_imp.R")
var_imp_overall <- var_imp(model_data)

# calculate partial dependence
source("scripts/partial_dependence.R")
pdp_data <- partial_dependence(model_data,var_imp_overall)
```

Note that the variable importance and partial dependence don't actually require direct inputs from the other functions above. However, the model selections for the variable importance are based off pieces of the earlier analysis. The very last step is to create plots from the paper:

```
# generate list of  plots
source("scripts/make_plots.R")
plots <- make_plots(all_preds,model_error,var_imp_overall,pdp_data)

# display the plots
plots$rmse_vs_unitrmse
plots$pred_vs_obs
plots$error_decomp
plots$var_imp
plots$partial_dep
```

[2]The monte carlo anaylsis requires 800 models for all 6 model types using loo-cv cross validation = over 100,000 models

## Extended analysis

This section includes more information about determining the hyperparameters of the models and the monte carlo analysis.

### Description of train_bl_models()

We classified multivariate regression and geostatistical models as "baseline" methods. This classification scheme is used as a way to compare groups of models and does not reflect the complexity, accuracy, or robustness of the method. The models include two versions of censored regression, ordinary kriging, and a unit-area discharge null model. The first thing to note is that the response varaible is slightly different for the censored regression models.

```
  model_data <- expVars$cleanBCs %>% # start with basin chars
    setNames(tolower(names(.))) %>% # make sure colname are lower case
    mutate(class = as.integer(as.factor(class))) %>% # change class to integer
    mutate(drain_sqkm = log(drain_sqkm)) %>%
    mutate_each(funs(as.numeric)) %>% # make everything numeric
    scale() %>% # convert to z-score: (x-mu)/sigma
    as.data.frame() %>% # make data frame
    mutate(y=data_full$y7q10+0.001) %>%
    mutate(y=log(y)) %>%
    select(y, class:aspect_eastness)
```

If you compare this setup to the `model_data` setup above, you will notice that the response variable is slightly different. This is because the the censored regression functions require a unique censoring value. In raw 7Q10 space, this is normally just zero. If we add a small constant to the 7Q10's and normalize them by their drainage basins, then there is no longer a unique censoring value (because the drainage basin areas are different). Therefore, we just take the natural log of the 7Q10 + 0.001 and censor as ln(0.001).

The only two models that require tuning are the two versions of the censored regression models. For example, the full censored regression model requires selecting a subset of predictors using best subset selection. It is impossible to know which predictors or the best selection algorithm to select apriori, so this is tunes using cross validation. The first step is to create a grid of possible values:

```
# tune parameter grid for lcr_tobit
  lcr_grid <- expand.grid(subset_method=c("forward","backward"),
                          subset_number=c(2,4,8,10,12))

print(lcr_grid)
```

```
##    subset_method subset_number
## 1        forward             2
## 2       backward             2
## 3        forward             4
## 4       backward             4
## 5        forward             8
## 6       backward             8
## 7        forward            10
## 8       backward            10
## 9        forward            12
## 10      backward            12
```

and in this case, 10 different models are built inside a loo-cv framework and the RMSE is recorded for each

model. The model with the lowest RMSE is selected as the final model. This process is referred to a "grid search".

```
  lcr_tune_error <- numeric()
  for (i in 1:nrow(lcr_grid)){

    # fit model with values from grid
    fit <- lcr.tobit(model_data,
                     subset_method=as.character(lcr_grid[i,1]),
                     subset_number=lcr_grid[i,2],
                     thold=log(0.001))

    # record rmse for ith model
    lcr_tune_error[i] <- sqrt(mean((fit$obs-fit$pred)^2,na.rm=T))
  }
```

A similar process is done for the region of influenced censored regression model, but with an additional "neighbors" hyperparameters.

```
# tune parameter grid for ROI lcr_tobit
  roi_grid <- expand.grid(neighbors=seq(25,200,25),
                          subset_method=c("forward","backward"),
                          subset_number=c(2,4,8,10,12))

  roi_tune_error <- numeric()
  for (i in 1:nrow(roi_grid)){

    # fit model with values from grid
    fit <- roi.lcr.tobit(model_data,
                         neighbors=roi_grid[i,1],
                         subset_method=as.character(roi_grid[i,2]),
                         subset_number=roi_grid[i,3],
                         thold=log(0.001))

    # record rmse for ith model
    roi_tune_error[i] <- sqrt(mean((fit$obs-fit$pred)^2,na.rm=T))
  }
```

### Description of train_ml_models()

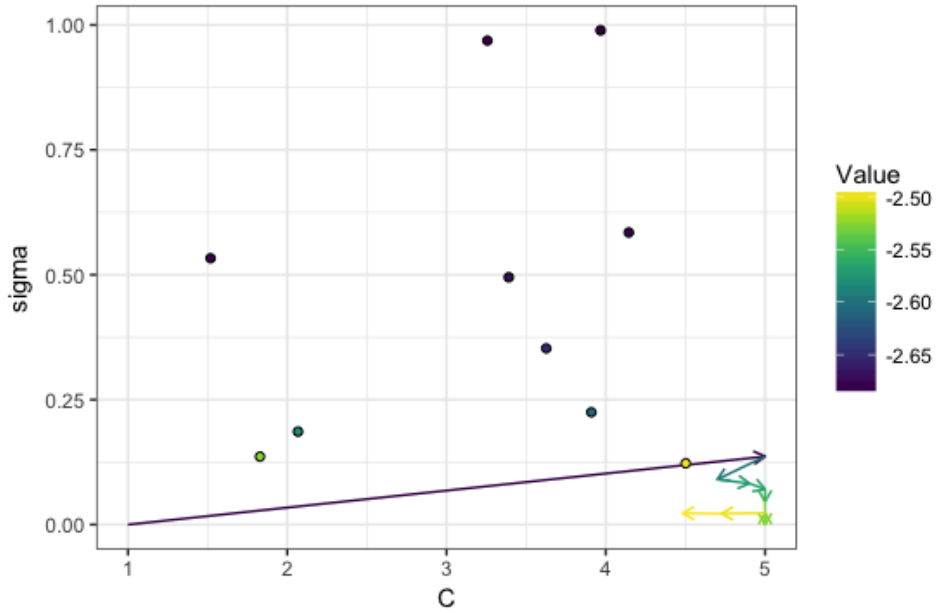
The process for the machine learning models is conceptually similar to the baseline models, but is much more computationally expensive. This basic pipeline involves (1) generating a bunch of points in hyperparameters and RMSE space to be used as substrate for a gaussian process model, and (2) use bayesian optimization to estimate the functional relationship between the hyperparameters and the RMSE. The gaussian process model attempts to discover a function that relates values of the hyperparameters to an output, in this case, the RMSE. Then an optimization function is used to seek for the combination of parameters where the function is maximized. Theoretically this should result in a set of optimal hyperparameters for a given model. Below is an example using a support vector machine (SVM) model. There are only two hyperparameters for a SVM model, the effect that the large residuals have on the regression is controlled by the *cost (C)* and the standard deviation of the Gaussian kernel is controlled by *sigma*. We don't know what values to select apriori, so we use the steps above. I have automated this process using a custom function that is a wrapper around both the `train` function from the caret package and the `BayesianOptimization` function from the rBayesianOptimization package.

```
library(caret)
source('scripts/bayes_optim_caret.R')

# list of parameter boundaries
svmg_bounds <-  list(C = c(1,5),
                     sigma = c(0,1))

# pass to custom function
svmg_params <- bayes_optim_caret(model_data,'svmRadial',svmg_bounds,iter=10,acq='ucb')
```

We can easily plot the parameter search for the SVM as it only has two parameters. The algorithm first seeds the search space with 10 random RMSE values resulting from different combinations of parameters. These are represented as the colored points below. The value is the negative RMSE in log space. We take the negation of the RMSE because the algorithm seeks to maximize the function. The line segments and the arrows represent the iterations of the Gaussian process model, with the color of the line indicating the resulting -RMSE for each iteration.




**Description of monte_carlo()**

The Monte-Carlo analysis attempts to answer the following questions: How does the predictive accuracy of a model compare relative to the other models as the amount of training data is increased? And similarly, would prediction accuracy be substantially improved with an increased number of gaged basins (i.e., are the models continuing to improve as we add more training data and is there evidence that installing more stream gages would be beneficial)? The Monte Carlo Analysis presented here can only answer the aforementioned questions relative to this particular dataset and these particular models. This is the algorithmic form of the monte carlo:

The "monte_carlo.R" function only completes a partial example using the k-nearest neighbors and null models. The partial analysis only using 10% and 20% of the data, and only repeats each step 10 times.

**Algorithm 1:** Number of Sites Monte Carlo Analysis

**Data**: 224 stream gages with 7Q10 values and basin characteristics

**Result**: Distribution of errors for variable number of sites

```
/* Local variables                                                    */
```

fraction $= [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]$;

n = length(fraction);

group1 = 7Q10 $>$ 90th percentile;

group2 = 7Q10 $=<$ 90th percentile;

**for** $i = 1$ **to** $n$ **do**

  **for** $j = 1$ **to** 100 **do**

    Randomly sample fraction[i] of rows from group 1;

    Randomly sample fraction[i] of rows from group 2;

    Combine samples;

    **for** $k = 1$ **to** 10 **do**

      Tune model parameters using LOO-CV for random subset ;

    **end**

    Calculate LOO-CV model error;

    Store model error[i, j];

  **end**

**end**