Defining `nested_spatial_cv()`: A Method for Spatial Nest Cross Validation for a Traditional Random Forest Model

Thesis for a Master of Public Health, Epidemiology

Nathan Garcia-Diaz

Brown University, School of Public Health

August 22, 2024

Contents

Note: the table of contents acts as in-document hyperlinks

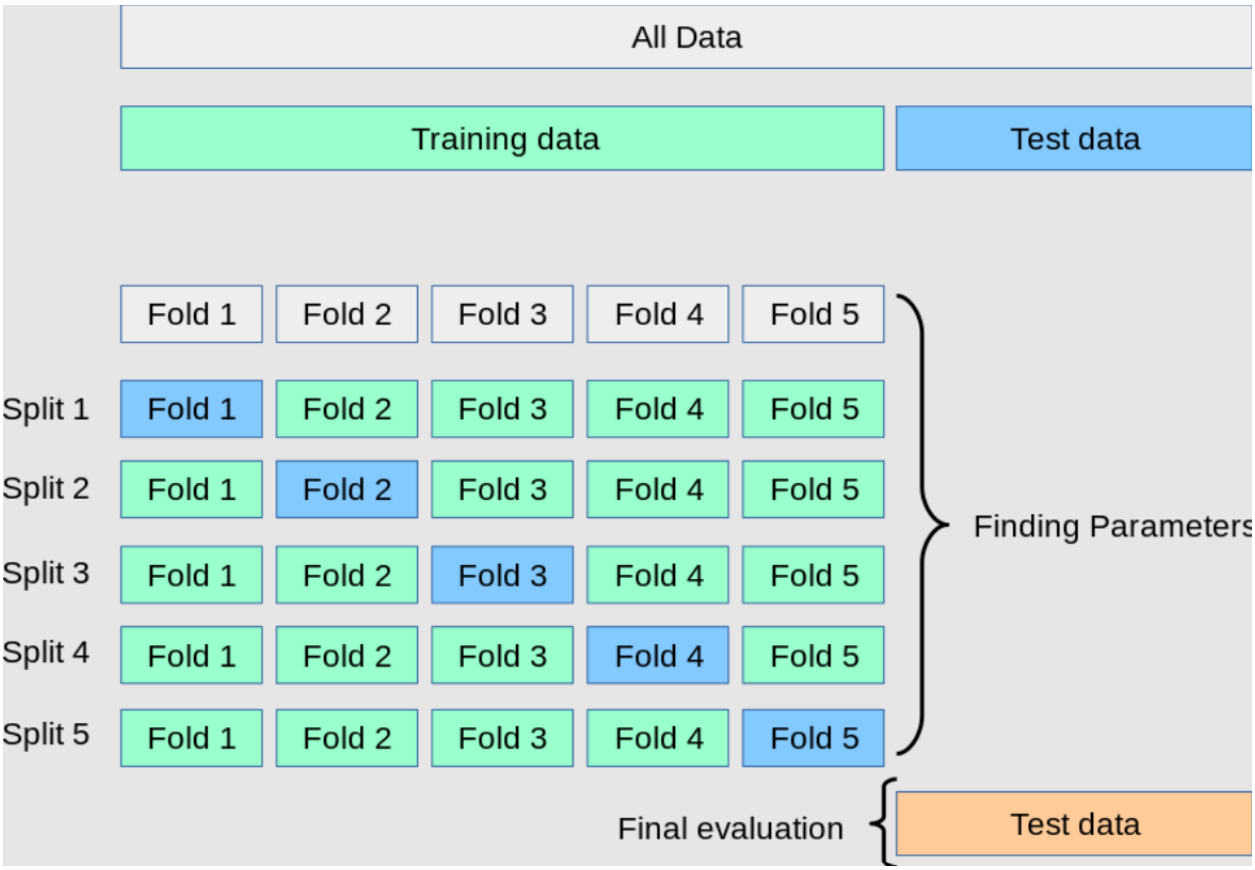
Purpose Statement

The following code describes the spatial nest-cross validation function used within the *Unveiling Vulnerability: Implementation of Geographically Weighted Random Forest Models across Federally Provided Indices*. Nested cross-validation is a technique used to assess the performance of a model while tuning hyperparameters. It helps to avoid over fitting and provides an unbiased estimate of model performance. A spatial nested cross-validation is a two-level cross-validation procedure designed to evaluate a model's performance and tune its hyperparameters simultaneously. `nested_spatial_cv()` can be considered to be spatial cross validation derived method because it manages a autocorrelation through the use of the `blockCV` package; whenever cross validation is mentioned it refers to a spatial cross-validation. The function involves two main stages:

- Outer Cross-Validation Loop:
  - Purpose: To estimate the model's performance on unseen data and provide a more reliable measure of how well the model generalizes to new data.
  - Procedure: The data set is divided into several folds (e.g., 5 or 10). In each iteration, one fold is used as the test set, and the remaining folds are used for training and hyper parameter tuning. This process is repeated for each fold, ensuring that every data point is used for testing exactly once.
- Inner Cross-Validation Loop:
  - Purpose: To select the best hyperparameters for the model.
  - Procedure: Within each training set from the outer loop, a further cross-validation is performed. This involves splitting the training data into additional folds (e.g., 3 or 5). The model is trained with various hyper parameter combinations on these inner folds, and the performance is evaluated to choose the optimal set of hyperparameters.

Example Workflow:

- Split the data into outer_k folds.
- For each fold in the outer loop:
- Use outer_k - 1 folds for training.
- Apply the inner cross-validation on this training set to tune hyperparameters.
- Evaluate the performance of the model with the selected hyperparameters on the held-out test fold.
- Average the performance metrics across all outer folds to get an overall estimate.


All Data
Training data
Test data
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 1
Split 2
Split 3
Split 4
Split 5
Finding Parameters
Final evaluation
Test data

Preparation

```r
# Load the libraries
library(sp)         # Spatial data handling
library(sf)         # Simple feature data handling
library(caret)      # Cross-validation and model tuning
library(dplyr)      # Data manipulation
library(cluster)    # Clustering methods
library(readr)      # Read a csv file
library(tigris)     # obtain shp files
library(randomForest) # Random Forest implementation
library(blockCV)
library(doParallel)
library(foreach)

# setting seed
set.seed(926)
# Load and prepare the spatial data
svi_df = read_csv(here::here("01_Data", "svi_df.csv")) %>%
  mutate(fips = as.character(fips)) %>%
  select(-...1) # if needed
# obtaining SPH files for RI tracts
tracts = tracts(state = "RI", year = 2022, cb = TRUE)
# joining data
map = inner_join(tracts, svi_df, by = c("GEOID" = "fips")) %>%
  mutate(fips = GEOID) %>%
  select(fips, rpl_themes, starts_with("e_"))
```

Defining Functions

2 functions are defined: `customRF()` and `nested_spatial_cv()`

`customRF()`: Overview

Overview: This function is used to the preform an exhaustive grid search of `ntree` and `mtry` random forest hyperparameters. This is used as a method for the `nested_spatial_cv_with_clustering()` function. This is code is provided by Jason Brownless (2020) at Tune Machine Learning Algorithms in R (random forest case study).

```
model <- train(
  response_var ~ .,  # Formula: specifies the response variable and all predictor variables
  data = df,  # Data: the training dataset containing both predictors and the response variabl
  method = customRF,  # Method: specifies the custom Random Forest model defined as 'customRF'
  trControl = trainControl(method = "cv", number = 5),  # trControl: controls the training pro
  tuneGrid = tuning_grid <- expand.grid(
  .mtry = c(2, 4, 6),
  .ntree = c(100, 200, 500))
)
```

Explanation: `customRF()` is a custom Random Forest model for exhaustive grid search with caret

```
customRF <- list(
   # Define the model type as regression (since Random Forest can be used for both classificat
  type = "Regression",
  # Specify the package required for the model
  library = "randomForest",
  # Not used in this example, but caret uses it for looping through models with different para
  loop = NULL
)

# Define the hyperparameters to tune (mtry and ntree) and their data types
customRF$parameters <- data.frame(
  parameter = c("mtry", "ntree"),  # Hyperparameters to be tuned
  class = rep("numeric", 2),  # Data types for the hyperparameters
  label = c("mtry", "ntree")  # Labels for the hyperparameters
)

# Grid function for hyperparameter tuning (left empty as we are not customizing the grid searc
customRF$grid <- function(x, y, len = NULL, search = "grid") {}

# Define the model fitting function using the Random Forest algorithm
customRF$fit <- function(x, y, wts, param, lev, last, weights, classProbs, ...) {
  randomForest(x, y, mtry = param$mtry, ntree = param$ntree, ...)  # Train the model with spec
}

# Define the prediction function for the model
customRF$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL) {
```

```
  predict(modelFit, newdata)  # Make predictions using the trained model
}

# Define a function for returning predicted probabilities (only applicable if the model suppor
customRF$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL) {
  predict(modelFit, newdata, type = "prob")  # Predict probabilities (useful in classification
}

# Define a function to sort the grid results based on mtry (used by caret)
customRF$sort <- function(x) x[order(x[, 1]),]

# Define a function to extract the levels (classes) in the model (only relevant for classifica
customRF$levels <- function(x) x$classes
```

nested_spatial_cv_with_clustering()

Overview: The code implements a nested cross-validation procedure specifically designed for spatial data. It begins by calculating spatial autocorrelation to comprehend spatial relationships within the dataset. Next, it creates spatially stratified folds for both outer and inner cross-validation. The custom Random Forest model is then trained and tuned within these spatial folds. Finally, the performance of the model is evaluated on both the inner and outer validation sets.

```
FUNCTION nested_spatial_cv_with_clustering(
    data,               # The dataset
    response_var,       # The response variable to predict
    spatial_col,        # The column with spatial information (default: "geometry")
    tuning_grid,        # The grid of hyperparameters for tuning
    outer_k,            # Number of outer folds
    inner_k,            # Number of inner folds
    metric              # Performance metric (default: "MSE")
)
```

Explanation: `nested_spatial_cv()` is a custom function that, and the following is the pseudo code

- Step 1: Calculate the spatial autocorrelation range using a helper function.
- Step 2: Create spatially stratified outer folds using the calculated range.
- Step 3: Set up parallel processing to speed up the nested cross-validation process.
- Step 4: For each outer fold:
  - Split the data into training and testing sets.
  - For each inner fold within the training data:
  - Split the inner training data further.
  - Train the model on the inner training data.
  - Evaluate the model's performance on the inner validation data.
  - Select the best model from the inner folds.
  - Evaluate the selected model on the outer testing data.
- Step 5: Stop parallel processing once all folds are processed.
- Step 6: Return the results of the cross-validation.

```r
# Function to calculate spatial autocorrelation range
calculate_autocor_range <- function(data, spatial_col) {
  autocor_range <- blockCV::cv_spatial_autocor(data[[spatial_col]])$range
  return(autocor_range)
}

# Function to create outer folds
create_outer_folds <- function(data, outer_k, autocor_range, spatial_col) {
  outer_folds <- blockCV::cv_spatial(data, k = outer_k, range = autocor_range, spatial_col = sp
  return(outer_folds)
}

# Function to create inner folds
create_inner_folds <- function(training_data, inner_k, autocor_range, spatial_col) {
  inner_folds <- blockCV::cv_spatial(training_data, k = inner_k, range = autocor_range, spatial
  return(inner_folds)
}

# Function to train the model
train_model <- function(inner_training_data, response_var, tuning_grid, inner_k) {
  model <- caret::train(
    as.formula(paste(response_var, "~ .")),  # Formula using the response variable
    data = inner_training_data,
    method = customRF,  # Method for training (custom Random Forest)
    trControl = caret::trainControl(method = "cv", number = inner_k),  # Inner cross-validatio
    tuneGrid = tuning_grid  # Grid of hyperparameters to tune
  )
  return(model)
}

# Function to evaluate the model
evaluate_model <- function(model, validation_data, metric, response_var) {
  predictions <- predict(model, newdata = validation_data)
  true_values <- validation_data[[response_var]]

  if (is.factor(true_values)) {
    # Classification: Calculate accuracy
    performance_metric <- mean(predictions == true_values)
  } else {
    # Regression: Calculate specified metric
    if (metric == "MSE") {
      performance_metric <- mean((predictions - true_values)^2)
    } else if (metric == "OOBMSE") {
      performance_metric <- model$finalModel$mse[model$bestTune$ntree]
    } else if (metric == "MAE") {
      performance_metric <- mean(abs(predictions - true_values))
    }
```

```r
  }

  return(performance_metric)
}

# Main function for nested spatial cross-validation with hierarchical clustering
nested_spatial_cv_with_clustering <- function(data,
                                              response_var,
                                              spatial_col = "geometry",  # Add spatial_col arg
                                              tuning_grid = expand.grid(mtry = c(3:8),
                                                                        ntree = c(250, 300, 350
                                              outer_k = 5,
                                              inner_k = 5,
                                              metric = "MSE") {

  # Calculate the spatial autocorrelation range
  autocor_range <- calculate_autocor_range(data, spatial_col)

  # Create outer folds
  outer_folds <- create_outer_folds(data, outer_k, autocor_range, spatial_col)

  # Initialize parallel processing to speed up computation
  cl <- parallel::makeCluster(parallel::detectCores())
  doParallel::registerDoParallel(cl)

  # Perform nested cross-validation
  outer_results <- foreach::foreach(i = seq_len(outer_k), .combine = 'c') %dopar% {
    # Access the indices for the current outer fold
    fold_indices <- outer_folds[[i]]

    # Split the data into training and testing based on the current outer fold
    training_data <- data[-fold_indices, ]
    testing_data <- data[fold_indices, ]

    # Create inner folds for hyperparameter tuning within the training data
    inner_folds <- create_inner_folds(training_data, inner_k, autocor_range, spatial_col)

    # Perform hyperparameter tuning using inner cross-validation
    inner_results <- lapply(seq_len(inner_k), function(j) {
      inner_fold_indices <- inner_folds[[j]]
      inner_training_data <- training_data[-inner_fold_indices, ]
      inner_validation_data <- training_data[inner_fold_indices, ]

      # Train the model on the inner training data
      model <- train_model(inner_training_data, response_var, tuning_grid, inner_k)

      # Evaluate the model on the inner validation data
```

```
      performance_metric <- evaluate_model(model, inner_validation_data, metric, response_var)

      return(list(
        fold = j,
        model = model,
        performance_metric = performance_metric
      ))
    })

    # Select the best model from the inner cross-validation results
    best_inner_model <- inner_results[[which.min(sapply(inner_results, function(res) res$perfo

    # Evaluate the best inner model on the outer testing data
    outer_performance_metric <- evaluate_model(best_inner_model, testing_data, metric, response

    return(list(
      fold = i,
      best_inner_model = best_inner_model,
      outer_performance_metric = outer_performance_metric
    ))
  }

  # Stop parallel processing and release resources
  parallel::stopCluster(cl)

  return(outer_results)
}
```