Predicting Civilian Involvement in Conflicts in Haiti

Joseph David

June 8 2022

Abstract

In this project, we use data on Haitian internal conflicts to predict the involvement of civilians in said conflicts. The three predictors we use are location, number of fatalities, and actors involved (other than civilian). We use Nadaraya-Watson regression, regression trees, and boosting with trees. We found that none of these are very strong predictors, but actor type was strongest followed by number of fatalities.

1 Introduction and Background

Nonparametric regression is a form of regression where the learned function comes solely from the information of the observed data, as opposed to having a predetermined form dependent on a set of parameters. Thus, nonparametric regression can be a useful tool when one does not wish to impose assumptions on the form of the model, and can be applied in a wide variety of cases of data.

In this project, we use a variety of nonparametric regression techniques such as Nadaraya-Watson estimators, decision trees, and boosting on a specific set of data from the Armed Conflict and Event Data Project (see the Dataset Description section below). The main features of interest or the position of events (latitude/longitude), the actors involved (eg. civilians, state forces, militias, etc.), as well as number of fatalities.

First, we begin by using Nadaraya-Watson regression in order to understand the effects of event position on number of fatalities. Then, we use classification trees to determine how well event position, fatalities, and actor type predict whether civilians were present at any given event. Finally, we use boosting to do something idk.

2 Dataset Description

We use data from the Armed Conflict Location and Event Data Project (ACLED). The ACLED records data from violent conflicts around the world such as the location, date, number of fatalities, etc, going back to 1997. The data also includes what type of conflict occurred, such as a battle, protest, or riot, as well the types of actors involved such as state forces, civilians, and protesters.

Our data, in particular, will consist of events that occurred in Haiti beginning January 1, 2021 and up until the data was obtained, on May 22, 2022. This decision was made because the dataset is so large that it was infeasible to run analyses on all the data. Therefore, we restrict ourselves in both time and space. This may limit the questions we are able to ask.

3 Statistical Methods and Implementation

3.1 Nadaraya-Watson Regression

We begin by regressing the effect of event position (latitude/longitude) on the number of fatalities at any given event using Nadaraya-Watson (NW) with a Gaussian kernel. The NW regression method can be described as follows: Given our dataset $\{(x_i, y_i)\}_{i=1}^N$, produced by $y_i = f(x_i) + \epsilon_i$, and given a point x, we would like to form an estimate $\widehat{y}$ of $y = f(x)$ using our data. Given data points x_i that are "near" x, we would expect that y is near y_i, so we define $\widehat{y}$ by the weighted average

$$\widehat{y} = \frac{\sum_{i=1}^N k(|x - x_i|) \cdot y_i}{\sum_{i=1}^N k(|x - x_i|)}.$$

The function $k(\cdot)$ is called the kernel. It is chosen such that, since we would like closer points to x to be weighted more, we want $k(z)$ to obtain its max at $z = 0$ and to decay monotonically (on either side) to zero as $|z| \to \infty$. The rate at which the kernel decays is an important hyperparameter called the bandwidth. It controls, roughly, what we consider "close by" and "far away" points to be. In our case, since we are regressing on latitude and longitude, we use the 2D Gaussian kernel

$$k(u, v) = \frac{1}{2\pi\sigma^2} \exp\left\{ - \frac{u^2 + v^2}{2\sigma^2} \right\},$$

and choose bandwidths $= 0.1, 0.2$, and 0.5 since these represent reasonably close distances in terms of latitude/longitude coordinates.

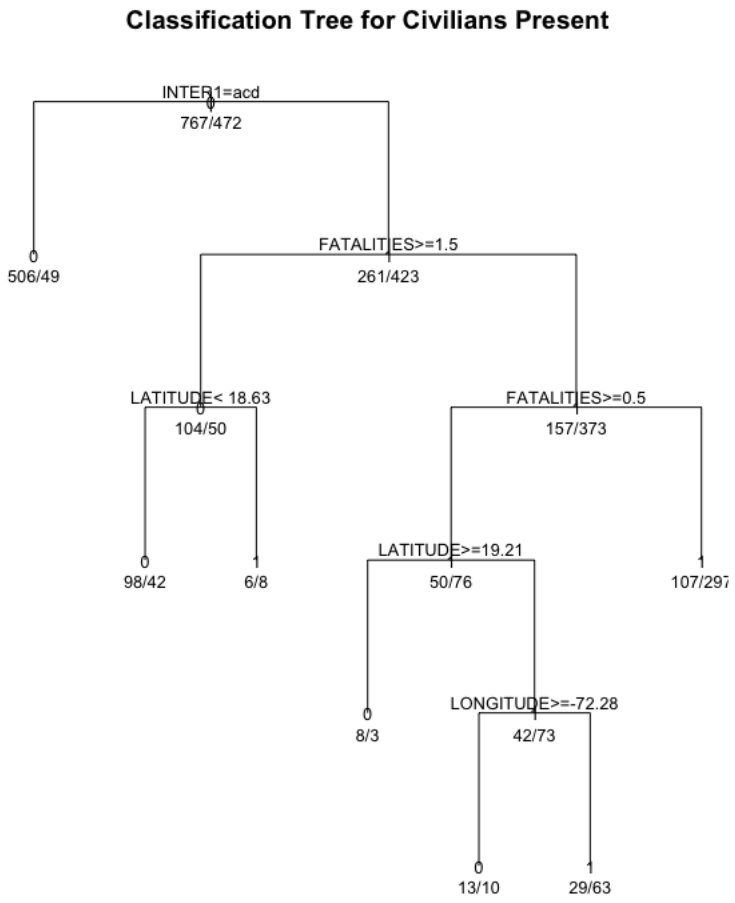
To implement NW regression, we wrote a function that takes a latitude/longitude pair as input and returns the weighted average estimate using a Gaussian kernel and given a preselected bandwidth. See the algorithm below.

Algorithm 1: Nadaraya-Watson Regression

Let x = longitude and y = latitude. Data is $\{(x_i, y_i)\}_{i=1}^N$.
Set bandwidth b, set $S = 0$.
for $i = 1, \ldots, N$ **do**
  Let $d = \sqrt{(x - x_i)^2 + (y - y_i)^2}$.
  Set $\widehat{y}_i = y_i *$ `dnorm(d, mean=0,sd=b).`
  $S = S +$`dnorm(d, mean=0,sd=b).`
end for
Return $\frac{\sum_{i=1}^N \widehat{y}_i}{S}$.

3.2 Regression Trees and Bagging

Next, we investigate whether event position, number of fatalities, or actor types can be used to predict whether civilians were present. In order to do this, we first had to create a "Civilians Present" feature in our dataset, which was possible since the dataset includes which actors are present. Since civilians are one of the actor types, we had to remove it from the list of possible actors since that would lead to trivial predictions. Below we fit a tree to the entire tree.



The most significant branch is on INTER1=*acd*. This variable corresponds to actor type, so the most significant indicator that civilians are not present is when state forces, political militias, or identity militias are involved in the conflict. Second to actor type, we have the number of fatalities. In particular, more fatalities implies that civilians are less likely to be present, given that the actor type is not one of the above mentioned. In particular, the right-most leaf node indicates that having FATALITIES $<= 0.5$) (i.e. no fatalities) is a significant indicator of civilians being present. Therefore, it appears that civilians present is correlated to low fatality and the other actors being protestors, rioters, and rebel forces.

We use the bagging method in order to test the prediction accuracy. The bagging method may be described as follows: One samples from the data $\{(x_i, y_i)\}_{i=1}^{N}$ as in bootstrapping, and fits a classification tree to this sample. We perform this B times to get the set of trees $\{\widehat{f}^b\}_{b=1}^{B}$. Then, we define a prediction model $\widehat{f}_{bag}$ for any point x by

$$\widehat{f}_{bag}(x) = \frac{1}{B}\sum_{i=1}^{N} \widehat{f}^b(x).$$

In our case, since the predictions will simply be a zero or one depending on whether there are civilians present or not, we round the average to the nearest integer.

Algorithm 2: Bagging Trees

Set number of bootstraps B.
for $b = 1, \ldots, B$ **do**
  Bootstrap sample (X', Y') from original data.
  Fit a tree $\widehat{f}^b(x)$ to (X', Y').
end for
Return $\widehat{f}(x) = \frac{1}{B} \sum_{b=1}^{B} \widehat{f}^b(x)$.

3.3 Boosting

Finally, we use the method of Boosting Regression Trres to supplement the investigation into whether event position, actor type, and number of fatalities can be used to predict whether civilians were present. Thus, we reuse the data from the Classification Trees section for the "Civilians Present" column.

Boosting is a method for finding slowly learning a regression model by iteratively fitting (usually simple) models to the data, taking the residuals of said model, fitting a model to the residuals, and so on. In our case, we are going to fit regression trees as in the previous section. Additionally, we will use cross validation in order to find the optimal hyperparameters for λ and B.

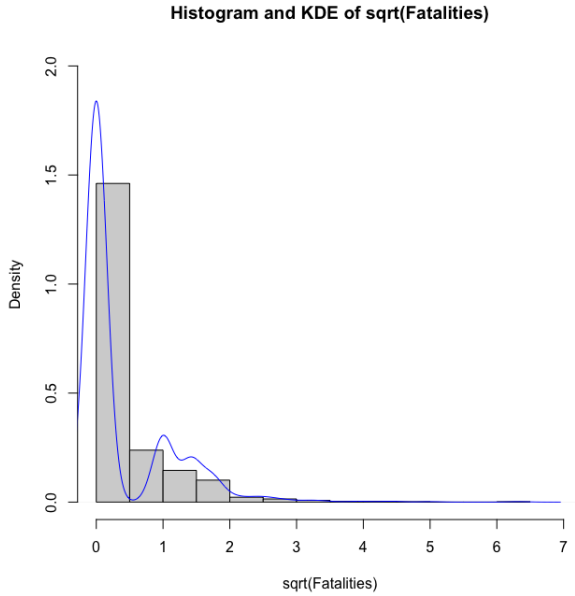
Algorithm 3: Boosting Regression Trees

Set $\widehat{f}(x) \equiv 0$. Let X be the covariates and $R = Y$ the target.
Set shrinkage parameter λ and number of boosts B.
for $b = 1, \ldots, B$ **do**
  Fit a tree $\widehat{f}^b(x)$ to the data (X, R).
  $\widehat{f}(x) = \widehat{f}(x) + \lambda * \widehat{f}^b(x)$.
  $R = R - \lambda * \widehat{f}^b(x)$.
end for
Return $\widehat{f}(x)$.

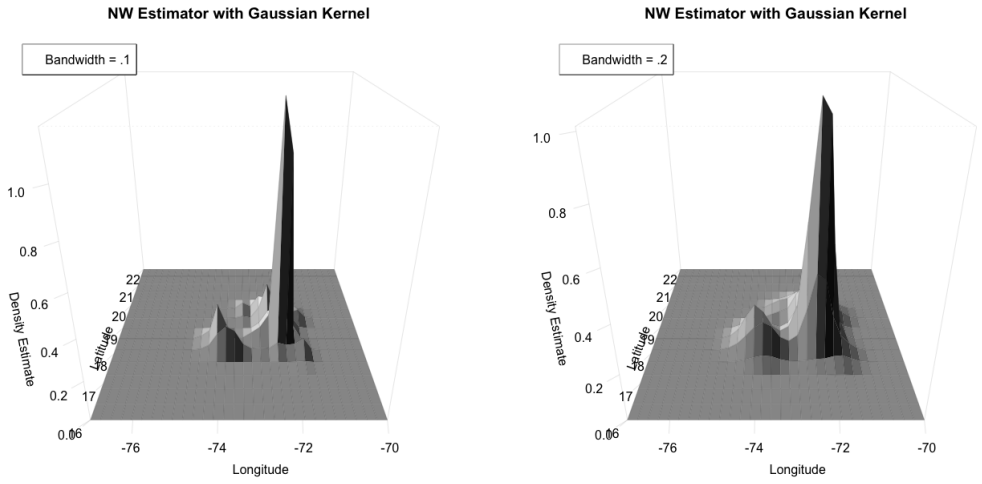
4 Results

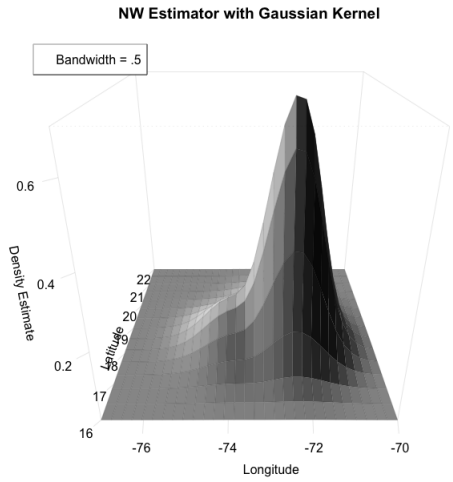
4.1 Nadaraya-Watson Regression

To get an idea of the data, we produce a histogram and kernel density estimate on the square root of the number of fatalities.



Here, we see that, by far, most events have zero fatalities, and, among the events that do have fatalities, the number is very small. In other words, the data is very skewed towards zero. Below are the NW regression functions for various bandwidths.





We see that, at its maximum, there is a predicted value of about one fatality, which reflects what we saw in the histogram. This one point also indicates that a singular event with many fatalities caused this spike.

Thus, we suspect that if more data was used, we would see a very different regression model. Since data from only one year was used, it seems that this outlier observation heavily skewed the results, so it is likely that these models are not useful.

4.2 Regression Trees and Bagging

In section 3.2 we saw that regression tree fit to the whole data, which showed that actor type and number of fatalities were the first predictors used. Below is the CP table for this tree.

```
Classification tree:
rpart(formula = CIVILIANS ~ LATITUDE + LONGITUDE + FATALITIES +
    INTER1, data = data.tree, subset = setdiff(1:1245, civilians.inter1),
    method = "class", control = rpart.control(minsplit = 20,
        minbucket = 10, cp = 0.004))

Variables actually used in tree construction:
[1] FATALITIES INTER1     LATITUDE   LONGITUDE

Root node error: 472/1239 = 0.38095

n= 1239

         CP nsplit rel error  xerror     xstd
1 0.3432203      0   1.00000 1.00000 0.036215
2 0.1144068      1   0.65678 0.65890 0.032335
3 0.0052966      2   0.54237 0.54449 0.030237
4 0.0042373      5   0.52542 0.58686 0.031071
5 0.0040000      6   0.52119 0.57839 0.030910
```

We see that actor type does a good job of reducing error, and all levels after that only make a small difference. However, the overall error is still not much better than a naive regression model.

Next, we tried to improve upon this by using bagging. We used the `R` function `rpart` in order to fit each tree. We created bagged trees with number of bootstrap samples $1, 2, 3, 4, 5, 10, \ldots, 95, 100$ and then calculated the classification error of the bagged trees on both the training data and on test data. Below are the results.

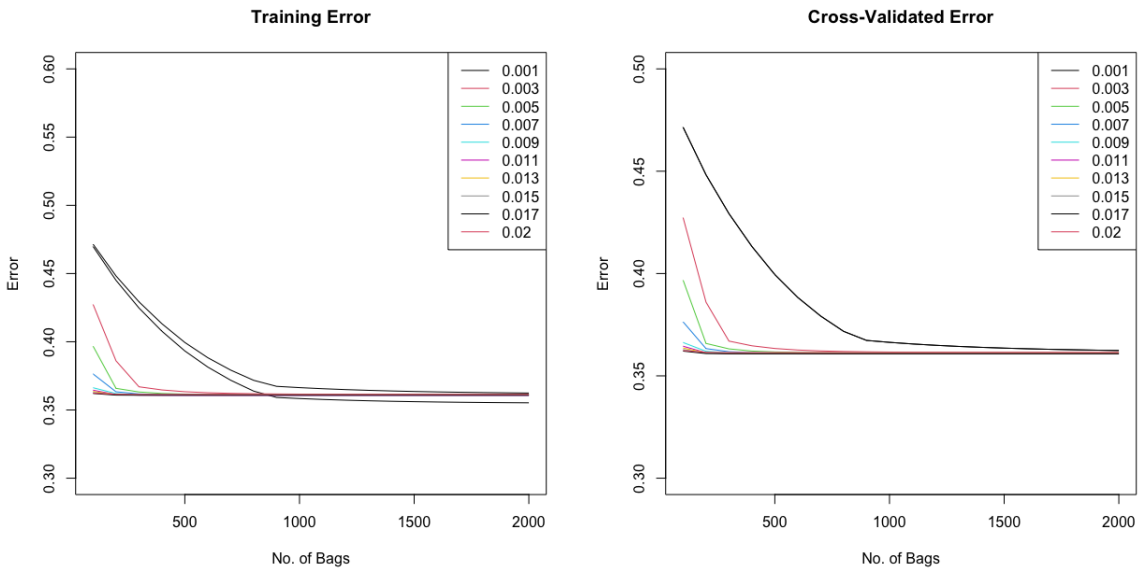
These results show that bagging did not have much of an effect on the classification error. Furthermore, the data had 480 observations out of a total of 1241 observations with civilians present, so a naive prediction model would obtain a $480/1241 = 37.8\%$ classification error. The bagged trees would not perform better than 20%, which shows some improvement but perhaps not very significant. We conclude that actor type and fatalities are a good predictor of civilians present, but event position is not.

4.3 Boosting

First, we split our dataset into training, cross-validation, and test sets. We tested all combinations of hyperparameters for

$$\lambda = .001, .003, \ldots, .02, \quad B = 100, 200, \ldots, 2000.$$

Below are the resulting training and cross validation mean square errors found.



We see that larger values of λ appear to perform better, but with diminishing returns. Furthermore, based on the training error, we see that, even with a large number of bags, we do not see much evidence of overfitting. However, we also see that after around 500 bags, the performance of boosting stagnates.

Based on the cross validation, we fit a boosted regression tree with $\lambda = 0.12$ and $B = 750$ and found that it had a classification error of 0.2. Note that this is only slightly better than the lower bound we saw in the Bagging section, so it appears that in this case boosting performed about as well as boosting, which is not what we would have expected.

5 Discussion and Conclusions

In conclusion, we found that the data given in the ACLED datasets does not hold significant predictive power in regards to whether civilians were involved in a given conflict. Given the variables of location, actor type, and fatalities, one might guess that this would have been the case, but now we have seen more evidence of this fact. However, we found that, as one would expect, actor type and number of fatalities, had higher predictive power than the event location. In particular, we found that state force, political and identity militia involvement was less correlated with civilian involvement, and that a lower number of fatalities was more correlated.

One limitation of this study is that the data was limited to a single year. Though there were still 1,250 observations, one might expect to see different results if more data was used. In particular, the effect of a few outlier observations in the Nadaraya-Watson regression seemed to skew the results, and so we suspect that the regression model is not particularly valid.

6 Code

```
#Stat527 Project
library(kdensity)
library(logKDE)
library(rpart)

data = read_excel('/Users/joseph/Courses/stat527/project/data.xlsx')

#Only take data from 2021-2022
data = data[data$EVENT_DATE > '2021-01-01',]
data = data[data$COUNTRY=='Haiti',]
n = length(data$ISO) #number of observations

#plot histogram of fatalities
hist(sqrt(data$FATALITIES), freq = FALSE, ylim=c(0,2), xlim=c(0,7),
     main='Histogram and KDE of sqrt(Fatalities)', xlab = 'sqrt(Fatalities)')

kde = kdensity(sqrt(data$FATALITIES), kernel = "gaussian")
lines(kde, main = 'KDE of Fatalities', col = 'blue')

kde.civilians = kdensity(c(sqrt(data$FATALITIES[data$INTER1==7]),
     sqrt(data$FATALITIES[data$INTER2==7])), kernel='gaussian')
plot(kde.civilians, main = 'KDE of Fatalities With/Without Civilians',
     col = 'red', xlab = 'sqrt(Fatalities)')
noncivil1 = which(data$INTER1!=7)
noncivil2 = which(data$INTER2!=7)
kde.noncivil = kdensity(sqrt(data$FATALITIES[intersect(noncivil1,noncivil2)]),
                        kernel = "gaussian")
lines(kde.noncivil, col = 'blue')
legend('topright', legend=c('Civilians Present','Civilians Not Present'),
       col = c('red','blue'), lty=c(1,1))

#NW Regression on Lat/Long, lower bandwidth
bandwidth = 0.2
new_long <- seq(-77,-70, length = 30)
new_lat <- seq(16,22, length = 15)
reg = outer(new_lat, new_long, NW_estimator)
persp(new_long, new_lat, t(sqrt(reg)), main='NW Estimator with Gaussian Kernel',
      col = 'white', theta = 0, phi = 25, shade = 0.75,lwd=0.1,
      xlab = 'Longitude', ylab = 'Latitude',zlab='Density Estimate',
      axes = TRUE, ticktype = 'detailed') #plot regression function
legend('topleft', legend=c('Bandwidth = .5'))

NW_estimator = function(lat,long){
  estimate = 0
  denom = 0
```

```
  for(i in 1:n){
    dist = sqrt((lat - data$LATITUDE[i])^2 + (long - data$LONGITUDE[i])^2)
    estimate = estimate + data$FATALITIES[i]*dnorm(dist, mean=0,sd = bandwidth)
    denom = denom + dnorm(dist, mean=0,sd = bandwidth)
  }
  estimate = estimate / denom
}

#########################################################################
#Fit Tree Decision model on whether civilians present
data.tree = data
#remove when actors are inter 1 (only 7 observations)
civilians.inter1 = which(data$INTER1==7)
#Set actor variable as type factor
data.tree$INTER1 = as.factor(data$INTER1)

fit.tree = rpart(CIVILIANS ~ LATITUDE + LONGITUDE + FATALITIES + INTER1,
                 subset = setdiff(1:1251,civilians.inter1), method = 'class',
                 data = data.tree,
                 control=rpart.control(minsplit=20, minbucket=10, cp=0.01))
printcp(fit.tree)
plotcp(fit.tree)
summary(fit.tree)
plot(fit.tree, uniform=TRUE,
     main="Classification Tree for Civilians Present") #plot tree
text(fit.tree, use.n=TRUE, all=TRUE, cex=.8)

#Classification Trees with Bagging
bs.sizes = seq(5,100, by=5)
bs.sizes = c(c(1,2,3,4), bs.sizes)
bs.train.errors = numeric(length(bs.sizes))
bs.test.errors = numeric(length(bs.sizes))

testData = data.tree[c("LATITUDE", "LONGITUDE", "FATALITIES",
                       "INTER1", "CIVILIANS")]
trainData = data.tree[c("LATITUDE", "LONGITUDE", "FATALITIES",
                        "INTER1", "CIVILIANS")]
indices = sample(1:n, 100)
testData = testData[indices,]
trainData = trainData[-indices,]
y.test.true = data.tree$CIVILIANS[indices]
y.train.true = data.tree$CIVILIANS[-indices]
n.train = length(trainData$LATITUDE)
n.test = length(testData$LATITUDE)

for(k in 1:length(bs.sizes)){
  train.predictions = numeric(n.train)
  test.predictions = numeric(n.test)
```

```
  for(i in 1:bs.sizes[k]){
    #Sample rows and fit a classification tree
    indices = sample(1:n.train, 100)
    fit.tree = rpart(CIVILIANS ~ LATITUDE + LONGITUDE + FATALITIES + INTER1,
                     subset = indices, method = 'class', data = trainData,
                     control=rpart.control(minsplit=5, minbucket=3, cp=0.1))
    #Generate predictions with this tree
    test.predictions = test.predictions + (predict(fit.tree,
                                                   testData, type='vector') - 1)
    train.predictions = train.predictions + (predict(fit.tree,
                                                   trainData, type='vector') - 1)
  }
  #classify by most common pred
  test.predictions = round(test.predictions / bs.sizes[k])
  train.predictions = round(train.predictions / bs.sizes[k])
  bs.test.errors[k] = sum(abs(test.predictions - y.test.true))/n.test
  bs.train.errors[k] = sum(abs(train.predictions - y.train.true))/n.train
}

plot(bs.sizes, bs.train.errors, xlab='Number of Bootstrap Samples',
     ylab='Classification Error', col = 'blue',ylim =c(.19, .3),
     main='Test vs Training Error')
lines(bs.sizes, bs.train.errors, col = 'blue')
lines(bs.sizes, bs.test.errors, col = 'green')
legend('topright', legend=c('Train','Test'), col = c('blue','green'),lty=1)

#####################################################################
#Boosting
data.boost = data.tree[c("LATITUDE", "LONGITUDE", "FATALITIES",
                         "INTER1", "CIVILIANS")]
data.boost$CIVILIANS = 2*data.boost$CIVILIANS - 1
indices.xval = sample(1:n, 100)
indices.test = sample(setdiff(1:n, indices.xval), 100)
indices.train = setdiff(1:n, union(indices.test, indices.xval))

testData = data.boost[indices.test,]
xvalData = data.boost[indices.xval,]

#X-Validate to determine shrinkage param and no. of trees
shrinkage = seq(.001, .02, length = 10)
bag_sizes = seq(100, 2000, length = 20)

trainError = xvalError = array(0, dim = c(length(shrinkage), length(bag_sizes)))
n.train = length(indices.train)
n.xval = length(indices.xval)
n.test = length(indices.test)

y.train = data.boost$CIVILIANS[indices.train]
```

```
y.xval = data.boost$CIVILIANS[indices.xval]
y.test = data.boost$CIVILIANS[indices.test]

for(i in 1:length(shrinkage)){
  for(j in 1:length(bag_sizes)){
    pred.train = numeric(n.train)
    pred.xval = numeric(n.xval)
    trainData = data.boost[indices.train,]
    for(k in 1:bag_sizes[j]){
      fit.tree = rpart(CIVILIANS ~ LATITUDE + LONGITUDE + FATALITIES + INTER1,
                       data = trainData, control=rpart.control(minsplit=5,
                       minbucket=3, cp=0.07))

      new.pred = predict(fit.tree, trainData, type='vector')
      pred.train = pred.train + shrinkage[i]*new.pred
      new.xval = predict(fit.tree, xvalData, type='vector')
      pred.xval = pred.xval + shrinkage[i]*new.xval

      trainData$CIVILIANS = trainData$CIVILIANS - shrinkage[i]*new.pred
    }
    #pred.train = pred.train/abs(pred.train)
    #pred.xval = pred.xval/abs(pred.xval)

    trainError[i,j] = 0.5 * mean((pred.train - y.train)^2)
    xvalError[i,j] = 0.5 * mean((pred.xval - y.xval)^2)
  }
}

plot(bag_sizes, xvalError[1,], main = 'Cross-Validated Error', xlab='No. of Bags',
     ylab='Error', ylim = c(0.3, 0.5), type = 'l')
for(i in 1:length(shrinkage)){
  lines(bag_sizes, xvalError[i,], col=i)
}
legend('topright', legend = trunc(shrinkage*10^3)/10^3, col=1:10, lty=1)

plot(bag_sizes, trainError[1,], main = 'Training Error',
     xlab='No. of Bags',
     ylab='Error', ylim = c(0.3, 0.6), type = 'l')
for(i in 1:length(shrinkage)){
  lines(bag_sizes, xvalError[i,], col=i)
}
legend('topright', legend = trunc(shrinkage*10^3)/10^3, col=1:10, lty=1)

#Fit tree with best hyperparameters
shrinkage = .012
num_bags = 750
pred.test = numeric(n.test)
indices.train = union(indices.train, indices.xval)
```

```
trainData = data.boost[indices.train,]
for(k in num_bags){
  fit.tree = rpart(CIVILIANS ~ LATITUDE + LONGITUDE + FATALITIES + INTER1,
                   data = trainData, control=rpart.control(minsplit=5,
                                        minbucket=3, cp=0.07))

  new.pred = predict(fit.tree, trainData, type='vector')
  new.pred.test = predict(fit.tree, testData, type='vector')
  pred.test = pred.test + new.pred.test*shrinkage
  trainData$CIVILIANS = trainData$CIVILIANS - shrinkage*new.pred
}
pred.test = pred.test/abs(pred.test)
error = 0.5 * mean(abs(pred.test - y.test))
plot(0.5*abs(pred.test - y.test))
```