```
# Stage 1: Flood-event classifier

# Necessary library
library(caret)

# Import historical FPR dataset (monthly, 2005-2015)
FPR_dataset <- read.csv(file.choose(),header = TRUE)

# Import 2016 daily FPR dataset
dataset_2016 <- read.csv(file.choose(),header = TRUE)

# Subset the datasets with the necessary variables
dataset <-
as.data.frame(FPR_dataset[,c("fips","NumClaims","max_ratio","ratio_greater_0.2",
                             "ratio_greater_0.5","ratio_greater_1","ratio_greater_2")])
dataset_2016 <- as.data.frame(dataset_2016[,c("fips",
"date","NumClaims","max_ratio","ratio_greater_0.2",
                             "ratio_greater_0.5","ratio_greater_1","ratio_greater_2")])

# list of all counties
all_fips <- unique(dataset$fips)

# Compute the number of claims in each county (between 2005-2015)
hist_claims <- dataset %>%
  dplyr::select(fips, NumClaims) %>%
  dplyr::group_by(fips) %>%
  dplyr::summarise(NumClaims = sum(NumClaims))

# Identify the counties with zero claims during 2005-2015 from analysis
hist_claims <- as.data.frame(hist_claims)
zero_claim_counties <- hist_claims[hist_claims[,"NumClaims"] == 0 , "fips"]

#Eliminate counties with zero claims during 2005-2015 from analysis
all_fips <- all_fips[!(all_fips %in% zero_claim_counties)]

# Note: Two counties (fips 12011 and 12086) have claims in every month during
2005-2016, thus they will be excluded.
all_fips <- all_fips[!(all_fips %in% c(12011,12086))]

# Convert the number of claims to a binary variable
dataset$NumClaims <- ifelse(dataset$NumClaims >= 1,"claim","no_claim")
dataset$NumClaims <- as.factor(dataset$NumClaims)

# Write a function to find maximum value of each column
colMax <- function(data) sapply(data, max, na.rm = TRUE)

# Create a blank data-frame to summarize the results of applying the
classifier
results_2016 <- as.data.frame(NULL)

# For each county, build a classifier and apply it to its 2016 daily FPR data
to identify when a flood began in a given county
for (i in 1:length(all_fips)) {
```

```
  # Loop progress status
  if (i%%100 ==0) print(i/length(all_fips))

  # Subset the dataset for county i
  dataset_sub <- subset(dataset, fips %in% all_fips[i])

  # Eliminate county's fips before subsequent analysis
  dataset_sub <- dataset_sub[,-1]

  # Eliminate an FPR metric if it is a zero variable (in other word, the max
and min values are zero)
    # Temporarily remove the number of claims variable
    dataset_sub_temp <- dataset_sub[,2:6]
    # Identify the non-zero FPR metrics
    dataset_sub_temp <-
as.data.frame(dataset_sub_temp[,colMax(dataset_sub_temp) != 0])
    dataset_sub <- cbind(dataset_sub[,1:2],dataset_sub_temp)
    # Eliminate duplicate max_ratio created
    dataset_sub[,2] <- NULL

  # Subset 2016 daily dataset for county i and the same non-zero FPR metrics
  dataset_sub_2016 <- subset(dataset_2016,fips %in% all_fips[i])
  dataset_sub_2016 <- dataset_sub_2016[,c(names(dataset_sub),"date")]

  # Convert the number of claims to a binary variable
  dataset_sub_2016$NumClaims <- ifelse(dataset_sub_2016$NumClaims >=
1,"claim","no_claim")
  dataset_sub_2016$NumClaims <- as.factor(dataset_sub_2016$NumClaims)

  ## Modeling
  #Cross Validation with downsampling
  control=trainControl(method="repeatedcv",   number=5, repeats = 3, sampling
= "down")

  # Build the classifier: random forest model
  set.seed(1)
  modelRF_fips= train(NumClaims
~.,data=dataset_sub,method="rf",trControl=control)

  # Apply the classifier to 2016 data
  predict_fips <- predict(modelRF_fips, newdata = dataset_sub_2016[,-
ncol(dataset_sub_2016)])

  # Summarize the results
  results_2016_temp <- cbind(dataset_sub_2016, predict_fips)
  results_2016_temp$fips <- all_fips[i]

  # Combine with prior counties' results
  if (i==1) {results_2016 <-
results_2016_temp[,c("NumClaims","max_ratio","date","predict_fips","fips")]}
else
  {results_2016 <- rbind(results_2016,
results_2016_temp[,c("NumClaims","max_ratio","date","predict_fips","fips")])}
```

```
}


## Forecast flood events

# Create a blank data-frame to summarize the forecasted flood events
flood_events <- as.data.frame(NULL)

# Iteration counter
z=0

# Identify forecasted flood events (note: flood events are assumed to have a
4-day period)
for (i in 1:length(unique(results_2016$fips))) {

  # Loop progress status
  if (i%%100 ==0) print(i/length(unique(results_2016$fips)))

  # Consider one county at a time
  results_2016_sub <- subset(results_2016, fips %in% unique(results_2016$fips)
[i])

  # Subset the days that the classifier identified as a claim day
  results_2016_sub_claimday <- subset(results_2016_sub, predict_fips %in%
"claim")

  # If the classifier identified no claim days for this county, move to the
next county
  if (nrow(results_2016_sub_claimday) == 0) next

  # Sort the data based on the date
  results_2016_sub_claimday <-
results_2016_sub_claimday[order(results_2016_sub_claimday$date),]

  # Blank object to store the dates that are covered by prior flood events
  counted_days <- NULL

  # Select a claim day as the starting day of an event
  for (j in 1:nrow(results_2016_sub_claimday)) {

    # Move to the next day if this day is covered in the prior flood events
    if (results_2016_sub_claimday[j,"date"] %in% counted_days) next

    # Find the 4-day period
    dates <- as.character(seq(as.Date(results_2016_sub_claimday[j,"date"]),
as.Date(results_2016_sub_claimday[j,"date"])+3, by="+1 day"))

    # Subset the 4-day period from the 2016 daily data
    dataset_2016_sub <- subset(dataset_2016,fips %in%
unique(results_2016$fips)[i] & date %in% dates)

    # Add the flood event to the summary object
```

```
    z = z + 1 # new row for the flood event
    flood_events[z,"fips"] <- unique(results_2016$fips)[i] #record county fips
    flood_events[z,"start_date"] <- dates[1] #record the start date of the
event
    flood_events[z,"end_date"] <- dates[4] #record the end date of the event
    flood_events[z,"NumClaims"] <- sum(dataset_2016_sub$NumClaims) #sum the
number of claims during the event

    # Store the days of this flood event to avoid overlapping between the
events
    counted_days <- rbind(counted_days, dates)

    }

}

# Snapshot of flooding events
head(flood_events)
```

```
# Stage 2: Predicting number of claims

# Necessary libraries
library(caret)
library(pscl)

# Import data
NFIP_df <- read.csv(file.choose(),header = TRUE)

# Covert the coastal variable to categorical
NFIP_df$coastal <- as.factor(NFIP_df$coastal)

# Cross Validation (10-fold CV, repeated three times)
control=trainControl(method="repeatedcv",   number=10,  repeats=3, verboseIter
= TRUE)


## Model 0: Null Model
MAE(mean(NFIP_df$NumClaims_all), NFIP_df$NumClaims_all)
RMSE(mean(NFIP_df$NumClaims_all), NFIP_df$NumClaims_all)


## Model 1: CART model
set.seed(1234)
modelbagging=
train(NumClaims_all~.,data=NFIP_df,method="treebag",trControl=control,importance
= TRUE, preProcess = c("center", "scale"))

# Model summary
modelbagging

# Standard deviation of error metrics
sd(modelbagging[["resample"]][["MAE"]])
sd(modelbagging[["resample"]][["RMSE"]])


## Model 2: SVM model
set.seed(1234)
modelSVM=train(NumClaims_all~.,data=NFIP_df,method="svmRadial",trControl=control,importanc
= TRUE, preProcess = c("center", "scale"))

# Model summary
modelSVM

# Standard deviation of error metrics
sd(modelSVM[["resample"]][["MAE"]])
sd(modelSVM[["resample"]][["RMSE"]])


## Model 3:  RF model
set.seed(1234)
modelRF=train(NumClaims_all~.,data=NFIP_df,method="ranger",trControl=control,importance
= "impurity", preProcess = c("center", "scale"))
```

```
# Model summary
modelRF

# Standard deviation of error metrics
sd(modelRF[["resample"]][["MAE"]])
sd(modelRF[["resample"]][["RMSE"]])


## Model 4: ZINB model

#Standardize numerical independent variables
NFIP_df[ , -which(names(NFIP_df) %in% c("NumClaims_all","coastal"))] =
scale(NFIP_df[ , -which(names(NFIP_df) %in%
c("fips","NumClaims_all","coastal"))], center= TRUE, scale=FALSE)

# Blank data frame to store the model results of each iteration
modelZINB_perf<- as.data.frame(NULL)
k=1

# Model development with cross validation similar to RF model (i.e. the same
10-fold CV, repeated three times as RF model)
for (i in 1:3){
  for (j in 1:10){

    # Use the same training and testing subsets as the ones created by RF
model
    NFIP_df2 <- NFIP_df[modelRF[["control"]][["index"]]
[[paste("Fold",sprintf("%02d",j) ,".Rep",i, sep="")]],]

    # Run the ZINB model
    set.seed(1234)
    hurdle_model <- zeroinfl(NumClaims_all ~  max_ratio + ratio_greater_0.5 +
ratio_greater_1 + ratio_greater_2 +
                               devlow + coastal + penetration_rate + water,
                             data = NFIP_df2, dist = "negbin", link="probit")

    # Calculate the error metrics in each iteration
    NFIP_df2 <- NFIP_df[modelRF[["control"]][["indexOut"]]
[[paste("Resample",sprintf("%02d",k), sep="")]],]
    modelZINB_perf[k,"RMSE"] <-
RMSE(predict(hurdle_model,NFIP_df2),NFIP_df2$NumClaims_all)
    modelZINB_perf[k,"MAE"] <-
MAE(predict(hurdle_model,NFIP_df2),NFIP_df2$NumClaims_all)
    modelZINB_perf[k,"Rsquared"] <-
R2(predict(hurdle_model,NFIP_df2),NFIP_df2$NumClaims_all)
    modelZINB_perf[k,"Cor"] <-
cor(predict(hurdle_model,NFIP_df2),NFIP_df2$NumClaims_all)

    # Next iteration
    k=k+1

  }
}
```

```
# Eliminate row if error metrics could not be calculated (i.e. Inf)
modelZINB_perf <- modelZINB_perf[is.finite(rowSums(modelZINB_perf)),]

# Mean RMSE
mean(modelZINB_perf$RMSE)

# Mean MAE
mean(modelZINB_perf$MAE)

# Mean Rsquared
mean(modelZINB_perf$Rsquared)

# Mean Corr.
mean(modelZINB_perf$Cor)

# Standard deviation RMSE
sd(modelZINB_perf$RMSE)

# Standard deviation MAE
sd(modelZINB_perf$MAE)
```