Stor 390 dat

T

2024-12-10

```r
IAdat <- read.csv("C:/Users/hinto/OneDrive/Documents/STOR390/390_FinalPaper/IA_datcleaned.csv")
IAdat

#selecting variables
IAdat <- IAdat[c("damagedStateAbbreviation", "householdComposition", "residenceType", "floodInsurance",
#IAdat$floodInsurance <- as.factor(IAdat$floodInsurance)
#IAdat$floodInsurance <- as.factor(IAdat$floodInsurance)
#IAdat$accessFunctionalNeeds <- as.factor(IAdat$accessFunctionalNeeds)
#IAdat$destroyed <- as.factor(IAdat$destroyed)

IAdat <- IAdat %>%
  mutate(householdComposition = case_when(
    householdComposition == ">5" ~ 6,
    TRUE ~ as.numeric(householdComposition)
  ))

IAdat$rentalAssistanceEligible <- as.factor(IAdat$rentalAssistanceEligible)
IAdat$residenceType <- as.numeric(factor(IAdat$residenceType))

IAdat %>%
  rename(
    State =  damagedStateAbbreviation,
  )


str(IAdat)
```

```r
#1.1 Removing all rows with zeros in every column

IAdat <- IAdat[!apply(IAdat, 1, function(row) all(row == 0)),] #There are no extra rows that we need to
IAdat

#1.2 Cleaning the data of outliers AND REPLACING OUTLIERS IWTH COLUMN MEANS

outlier_fill <- function(c) {
  z_score <- (c - mean(c))/sd(c) #z-score

  outliers <- abs(z_score) > 3 #ids outliers

  c[outliers] <-mean(c[!outliers], na.rm = TRUE) #fills in outliers with column mean excluding outliers
  return(c)
}
```

```r
IAdat[c("repairAmount", "foundationDamageAmount", "roofDamageAmount")] <- lapply(IAdat[c("repairAmount"
IAdat_clean <- IAdat[, -c(1,10)]
#
```

```r
#2.1 Model 1: KNN

 set.seed(123)
 ran <- sample (1: nrow(IAdat_clean), 0.8 * nrow(IAdat_clean))
 summary(IAdat_clean)
 summary(IAdat_clean[ran,])

 normal <- function(x){
(x - min(x))/(max(x) - min(x)) #we use min-max normalization here, as the authors suggest
 }
 IAdat_norm <- as.data.frame(lapply(IAdat_clean[, c(6,7,8)], normal))
IAdat_norm <- cbind(IAdat_norm, IAdat_clean[, -c(6,7,8)])

#At this point, we need to convert all factors to numeric values
#IAdat_norm[,c(8,9,10)] <- as.numeric(factor(IAdat_norm[,c(7,8,9)], levels = c("0","1"))) #change level
summary(IAdat_clean)
summary(IAdat_norm)
#
#
IA_train <- IAdat_norm[ran,]
IA_test <- IAdat_norm[-ran,]
#
IA_target_category <- IAdat_norm[ran, 9]
IA_test_category <- IAdat_norm[-ran, 9]
#
# #2.2 Using Cross-validation to find the proper k
# #change code on this sect more og
#
# library(class)
# library(caret)
#
#
# # Set seed for reproducibility
# set.seed(123)
#
# # Specify the number of observations for training and testing
# n_train <- 500
# n_test <- 200
#
# # Create a stratified split (ensures balanced class distribution)
# split <- createDataPartition(IAdat_clean$rentalAssistanceEligible, p = n_train / nrow(IAdat_clean), l
#
# # Create training and testing sets
# train_set <- IAdat_norm[split, ]
# test_set <- IAdat_norm[-split, ][1:n_test, ]  # Limit the test set to n_test observations
#
# # Check the dimensions
# dim(train_set)
# dim(test_set)
```

```
# train <- IAdat_norm[ran,] #differentiating cross-val training and testing sets from knn so that we ca
# test <- IAdat_norm[-ran,]
#
#
# #Trying the Hammig distance in R
# # Load required libraries
# library(caret)
# library(class)
#
# # Define a custom KNN function using Hamming Distance
# knn_hamming <- function(train, test, cl, k) {
#   n_test <- nrow(test)
#   predictions <- vector("numeric", n_test)
#
#   # Loop over each test sample
#   for (i in 1:n_test) {
#     # Calculate Binary Distance from test sample to all training samples
#     distances <- apply(train, 1, function(train_sample) {
#       sum(test[i, ] != train_sample)  # Count number of mismatches (Binary distance)
#     })
#
#     # Get k-nearest neighbors (sorted by distance)
#     nearest_neighbors <- order(distances)[1:k]
#
#     # Majority vote to predict class of the test sample
#     neighbor_labels <- cl[nearest_neighbors]
#
#     # Ensure that neighbor_labels is a factor and get the most common class
#     most_common_label <- names(sort(table(neighbor_labels), decreasing = TRUE)[1])
#
#     # Assign the majority vote label to predictions
#     predictions[i] <- most_common_label
#   }
#
#   return(predictions)
# }
#
# #accuracy <- knn_hamming(train = train_set[, -9], test = test_set[, -9], cl = train_set[, 9], k =5)
#
# # Custom K-Fold Cross-Validation for knn_hamming
# cross_validate_knn_hamming <- function(data, target, k_folds = 3, k_neighbors = 5) {
#
#   # Create a vector to store accuracy values for each fold
#   accuracy_values <- vector("numeric", k_folds)
#
#   # Create a fold assignment (e.g., split data into k_folds)
#   folds <- sample(1:k_folds, nrow(data), replace = TRUE)
#
#   # Perform cross-validation
#   for (fold in 1:k_folds) {
#
#     # Create training and testing data for this fold
#     train_data <- data[folds != fold, ]
```

```
#     test_data <- data[folds == fold, ]
#
#     # Create corresponding labels
#     train_labels <- target[folds != fold]
#     test_labels <- target[folds == fold]
#
#     # Run the custom KNN with Hamming Distance
#     predictions <- knn_hamming(train_data, test_data, train_labels, k_neighbors)
#
#     # Calculate accuracy for this fold
#     accuracy_values[fold] <- mean(predictions == test_labels)
#   }
#
#   # Return the average accuracy across all folds
#   mean_accuracy <- mean(accuracy_values)
#   return(mean_accuracy)
# }
#
# cross_validate_knn_hamming(train = train_set[, -9], test = test_set[, -9], cl = train_set[, 9], k =5)
#
#

# set.seed(123)
# k_values <- 1:20000
# accuracy <- sapply(k_values,  function(k) {
#   pred <- knn(train[, -9], test[, -9], cl = train[, 9], k = k, prob = TRUE)
#   mean(pred == test[, 9])
# })
#
# accuracy
#
#
# #2.3 Completing KNN from cross-validated k value
#
# #KNN_IA <- knn(IA_train, IA_test, cl = IA_target_category, k = 11)
#
#
# cross_validate_knn_hamming <- function(data, target, k_folds = 3, k_neighbors = 5, n_train = 100, n_t
#
#   # Create a vector to store accuracy values for each fold
#   accuracy_values <- vector("numeric", k_folds)
#
#   # Create a fold assignment (e.g., split data into k_folds)
#   folds <- sample(1:k_folds, nrow(data), replace = TRUE)
#
#   # Perform cross-validation
#   for (fold in 1:k_folds) {
#
#     # Create training and testing data for this fold
#     train_data <- data[folds != fold, ]
#     test_data <- data[folds == fold, ]
#
#     # Shorten training and testing sets to specific sizes (n_train, n_test)
#     if (nrow(train_data) > n_train) {
```

```r
#       train_data <- train_data[1:n_train, ]
#     }
#     if (nrow(test_data) > n_test) {
#       test_data <- test_data[1:n_test, ]
#     }
#
#     # Create corresponding labels
#     train_labels <- target[folds != fold]
#     test_labels <- target[folds == fold]
#
#     # Shorten the labels as well to match the data size
#     if (length(train_labels) > n_train) {
#       train_labels <- train_labels[1:n_train]
#     }
#     if (length(test_labels) > n_test) {
#       test_labels <- test_labels[1:n_test]
#     }
#
#     # Run the custom KNN with Hamming Distance
#     predictions <- knn_hamming(train_data, test_data, train_labels, k_neighbors)
#
#     # Calculate accuracy for this fold
#     accuracy_values[fold] <- mean(predictions == test_labels)
#   }
#
#   # Return the average accuracy across all folds
#   mean_accuracy <- mean(accuracy_values)
#   return(mean_accuracy)
# }
#
# cross_validate_knn_hamming(IAdat_norm, IAdat_norm$rentalAssistanceEligible, k_folds = 3, k_neighbors
```

```r
#3 - Logistic regression -we'll want chi-squared comparison for feature importance here

#3.1 - Using AIC for selecting the best model (feature importance) with normalized data

logreg_all <- glm(rentalAssistanceEligible ~ ., data = IAdat_norm, family = binomial)
logreg_null <- glm(rentalAssistanceEligible ~ 1, data = IAdat_norm, family = binomial) #null

#stepwise selection (AIC)
logreg_stepwise <- step(logreg_null, scope = list(lower = logreg_null, upper = logreg_all), direction =
 summary(logreg_stepwise)

#Get the p-values for the final model

logreg_final <- glm(rentalAssistanceEligible ~ repairAmount + destroyed + householdComposition +
    accessFunctionalNeeds + foundationDamageAmount + residenceType +
    roofDamageAmount + floodInsurance, data = IAdat_norm, family = binomial)
summary(logreg_final) #we see that all variables are extremely significant in the final model

#3.2 We need to produce a confusion matrix for predictions!

#Predictions
```

```r
predicted_probs <- predict(logreg_final, type = "response")
predicted_classes <- ifelse(predicted_probs > 0.5, 1, 0)

# Evaluate performance
conf_matrix <- table(Actual = IAdat_norm$rentalAssistanceEligible, Predicted = predicted_classes)

#Sens, acc, and prec
accuracy_logreg <- mean(predicted_classes == IAdat_norm$rentalAssistanceEligible)
FP <- conf_matrix[1, 2]
FN <- conf_matrix[2, 1]
TP <- conf_matrix[2, 2]
TN <- conf_matrix[1, 1]


#predicted positives that are really positive
precision_logreg <- TP / (TP + FP)

#true positives that we do correctly predict
sensitivity_logreg <- TP / (TP + FN)

accuracy_logreg
precision_logreg
sensitivity_logreg
```

```r
#4-Decision Tree
#making the normalized data smaller so that we can run  cross-val easier
set.seed(123)
index <- sample(nrow(IAdat_norm), 200000)
IAdat_dt <- IAdat_norm[index, ]

#4.1 Decision tree
IA.tree <- rpart(rentalAssistanceEligible ~ repairAmount + destroyed + householdComposition + accessFun
  data = IAdat_dt,
  method = "class"
)
#visualize the initial tree
par(xpd = NA) # otherwise on some devices the text is clipped
plot(IA.tree)
text(IA.tree, pretty = 0)
IA_dt_init <- rpart.plot(IA.tree) #this is the visualization that will go in the paper
#4.2 Eval decision tree
#Confusion matrix
create_train_test <- function(data, size = 0.8, train = TRUE) {
    n_row = nrow(data)
    total_row = size * n_row
    train_sample <- 1: total_row
    if (train == TRUE) {
        return (data[train_sample, ])
    } else {
        return (data[-train_sample, ])
    }
}
data_train <- create_train_test(IAdat_dt, 0.8, train = TRUE)
```

```
data_test <- create_train_test(IAdat_dt, 0.8, train = FALSE)
```

```
#Tree function for cv
set.seed(123)
index <- sample(1:nrow(IAdat_dt), 0.8 * nrow(IAdat_dt))
train_data <- IAdat_dt[index, ]
test_data <- IAdat_dt[-index, ]
treemod_cv <- tree(rentalAssistanceEligible ~ ., data = train_data)
```

#Initial Decision tree

```
IA_dt_initneat <- plot(treemod_cv)
text(treemod_cv, pretty = 0)
#work on this visualization
```

```
#this prediction uses the rpart formula
  predict_unseen <-predict(IA.tree, data_test, type = 'class')
  table_mat <- table(data_test$rentalAssistanceEligible, predict_unseen)
  table_mat

#Cross-validation, using tree rather than rpart predictions
set.seed(123)
cv_tree <- cv.tree(treemod_cv, FUN = prune.misclass)
plot(cv_tree$size, cv_tree$dev, type = "b", xlab = "Tree Size", ylab = "Misclassification Error")
num <- cv_tree$size[which.min(cv_tree$dev)]
prune.tree = prune.misclass(treemod_cv, num)
plot(prune.tree)
text(prune.tree, pretty=0)
```

Figure X Pruned Final Tree for paper

```
prune <- plot(prune.tree)
text(prune.tree, pretty = 4)
```

```
#4.3 - Making predictions from pruned model
test_pred <- predict(prune.tree, test_data, type = "class")
conf_matrix <- table(Predicted = test_pred, Actual = test_data$rentalAssistanceEligible)
conf_matrix
accuracy_dt <- mean(test_pred == test_data$rentalAssistanceEligible)

FP <- conf_matrix[1, 2]
FN <- conf_matrix[2, 1]
TP <- conf_matrix[2, 2]
TN <- conf_matrix[1, 1]

#predicted positives that are really positive
precision_dt <- TP / (TP + FP)

#true positives that we do correctly predict
```

```r
sensitivity_dt <- TP / (TP + FN)

#4.4 Reporting the accuracy and confusion matrix

accuracy_dt
precision_dt
sensitivity_dt
```