HEP particles Classification

Mohamed Elashri

10/05/2019

```
# import the testing data
data <- readr::read_csv("https://www.dropbox.com/s/34gwx6e5mwkts2u/data.csv?dl=1")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Label = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
# Remove the X1 column, which is simply an index
data <- data %>%
  dplyr::select(-c("X1"))
```

Analysis I

```
# Create a validation set and a training set from data
set.seed(1)
#Find the sample size, which is 505 of data
smp_siz <- floor(0.8*nrow(data))
#Randomly finds rows equal to sample size and indices of those rows
data_ind <- sample(seq_len(nrow(data)),size = smp_siz)
#One dataset includes these randomly found rows, the other does not
data_train <- data[data_ind,]
data_val <- data[-data_ind,]
```

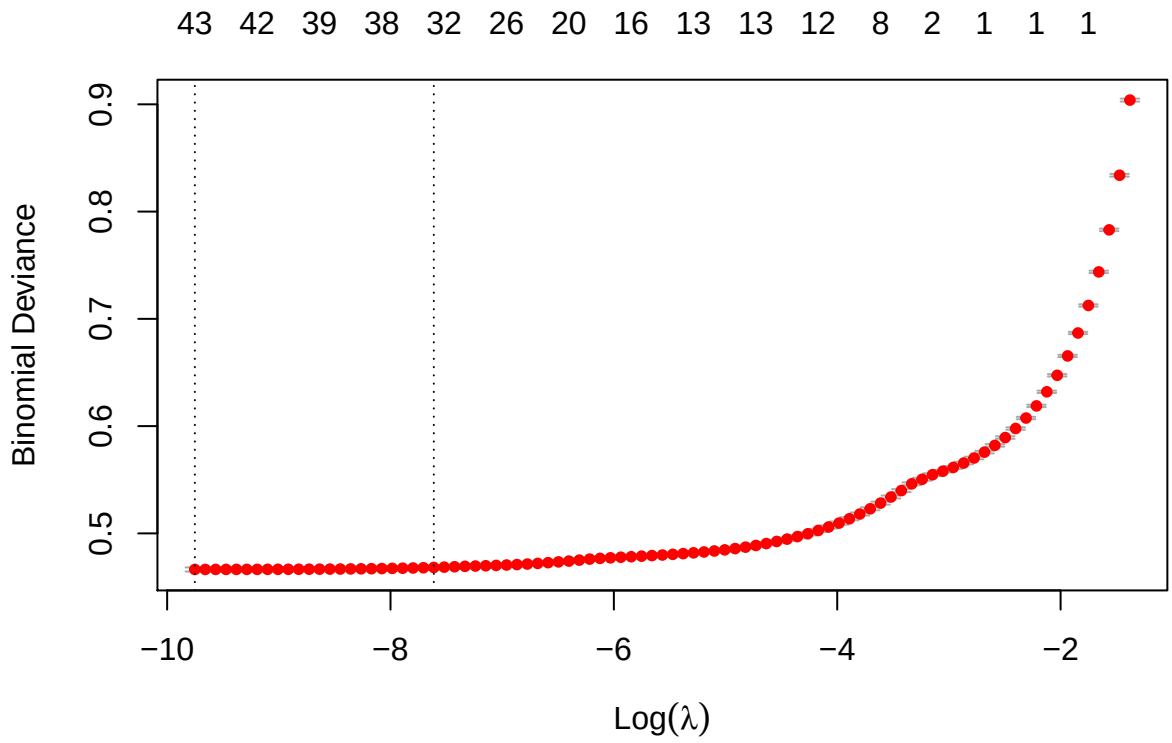
```
# Break up data into labels and predictors
train_labels <- data_train$Label
train_predictors <- data_train %>% dplyr::select(-c("Label"))
val_predictors <- data_val %>% dplyr::select(-c("Label"))
val_labels <- data_val$Label
# Filter out ghost particles and remove the label, as it should not be used as a predictor
no_ghost_train <- data_train %>% filter(Label != 'Ghost')
no_ghost_train_predictors <- no_ghost_train %>% dplyr::select(-c("Label"))
no_ghost_train_label <- no_ghost_train$Label
```

```
# Make a 0-1 Indicator if the particle is ghost or not
data_train %>% mutate(
  ghostLabel = ifelse(Label == "Ghost", 1, 0)
) -> ghost_train_df
# Format training data into form we can put into ml models
ghost_train_labels <- ghost_train_df$ghostLabel
ghost_train_df <- ghost_train_df %>% dplyr::select(-c("Label"))
```

```r
# Make a 0-1 Indicator if the particle is ghost or not
data_val %>% mutate(
  ghostLabel = ifelse(Label == "Ghost", 1, 0)
) -> ghost_val_df
# Format validation data into form we can put into ml models
ghost_val_df <- ghost_val_df %>% dplyr::select(-c("Label"))
```

```r
# Fit a logistic regression model using 10 fold cross validation
fit_cv2 <- cv.glmnet(as.matrix(train_predictors), ghost_train_labels, family = "binomial", nfolds = 10)
```

```r
# Visualize the result of the CV
plot(fit_cv2)
```




```r
# Print out the lambda value to minimize overfitting
fit_cv2$lambda.1se
```

```
## [1] 0.000494393
```

```r
# Get predicted probabilities and transform them into class values
probs <- predict(fit_cv2, as.matrix(val_predictors), s = "lambda.1se" , type = "response")
pred_cv <- ifelse(probs >= 0.5, 1, 0)
```

```r
u <- union(pred_cv, ghost_val_df$ghostLabel)
conf_matrix <- table(ordered(pred_cv, u, levels = c(0,1)), ordered(ghost_val_df$ghostLabel, u, levels =
#Generate a confusion matrix from the predicted and actual values
conf_mat <- caret::confusionMatrix(conf_matrix)$table
#Calculate sensitivity, specificity and total error
```

```r
sens <- (conf_mat[2,2]/colSums(conf_mat)[2])[[1]]
spec <- (conf_mat[1,1]/colSums(conf_mat)[1])[[1]]
total_err <- (conf_mat[1,2] + conf_mat[2,1]) / sum(conf_mat)
lr_fnp <- (conf_mat[1,2]/rowSums(conf_mat)[1])[[1]]
lr_fpp <- (conf_mat[2, 1]/rowSums(conf_mat)[2])[[1]]
lr_sens <- sens
lr_spec <- spec
lr_err <- total_err
# Create a table of model metrics
tbl_lr <- data.frame(lr_sens, lr_spec, lr_err, lr_fnp, lr_fpp)
tbl_lr
```

```
##     lr_sens   lr_spec     lr_err     lr_fnp    lr_fpp
## 1 0.6060743 0.9676645 0.09255833 0.07522957 0.2107252
```

```r
# Function to perform QDA and cross validation
calc_qda_vals <- function(data, tau_val, select_tau = FALSE) {
  set.seed(1)

  #Set k folds
  k <- 5


  #Cut the data into k folds
  folds <- cut(seq(1, nrow(data)), breaks = k, labels = FALSE)

  # Specify tau values to try for the thresholds
  if (select_tau) {
    taus <- seq(from = 0.05, to = 0.95, by = 0.05)

    # Initialize values
    avg_sens <- rep(0, times = length(taus))
    avg_spec <- rep(0, times = length(taus))
    avg_tot_err <- rep(0, times = length(taus))
  } else {

    # Initialize values
    avg_spec <- rep(0, times = k)
    avg_sens <- rep(0, times = k)
    avg_tot_err <- rep(0, times = k)
  }
  # Perform cross validation
  for(i in 1:k){

    #Segment data
      idx <- which(folds==i,arr.ind=TRUE)
      val <- data[idx, ]
      train <- data[-idx, ]


      #Fit the model
      fit <- MASS::qda(ghostLabel~.,data=train)

      #Find probabilities and predictions
```

```
    probs <- predict(fit, val)$posterior[,2]

    # Hyperparameter Search
    if (select_tau) {
      for(j in 1:length(taus)){
        param <- taus[j]
        pred <- ifelse(probs >= param, 1, 0)
        u <- union(pred, val$ghostLabel)
        conf_matrix <- table(ordered(pred, u, levels = c(0,1)),
                             ordered(val$ghostLabel, u, levels = c(0,1)))

        #Generate a confusion matrix from the predicted and actual values
        conf_mat <- caret::confusionMatrix(conf_matrix)$table

        #Calculate sensitivity, specificity and total error
        sens <- (conf_mat[2,2]/colSums(conf_mat)[2])[[1]]
        spec <- (conf_mat[1,1]/colSums(conf_mat)[1])[[1]]
        total_err <- (conf_mat[1,2] + conf_mat[2,1]) / sum(conf_mat)

        #Keep track of values for the current folds
        avg_spec[j] %+=% spec
        avg_sens[j] %+=% sens
        avg_tot_err[j] %+=% total_err
      }
    } else {

        # From threshold, get class values
        pred <- ifelse(probs >= tau_val, 1, 0)

        #Generate a confusion matrix from the predicted and actual values
        u <- union(pred, val$y)
        conf_matrix <- table(ordered(pred, u, levels = c(0,1)),
                             ordered(val$y, u, levels = c(0,1)))
        conf_mat <- caret::confusionMatrix(conf_matrix)$table

        #Calculate sensitivity, specificity and total error
        sens <- (conf_mat[2,2]/colSums(conf_mat)[2])[[1]]
        spec <- (conf_mat[1,1]/colSums(conf_mat)[1])[[1]]
        total_err <- (conf_mat[1,2] + conf_mat[2,1]) / sum(conf_mat)

        #Keep track of values for the current folds
        avg_spec[i] = spec
        avg_sens[i] = sens
        avg_tot_err[i] = total_err

    }
}

if (select_tau) {

  # Calculate metrics
  tot_err <- avg_tot_err/k
  spec_sens_calc <- (avg_spec/k + avg_sens/k )/ 2
```

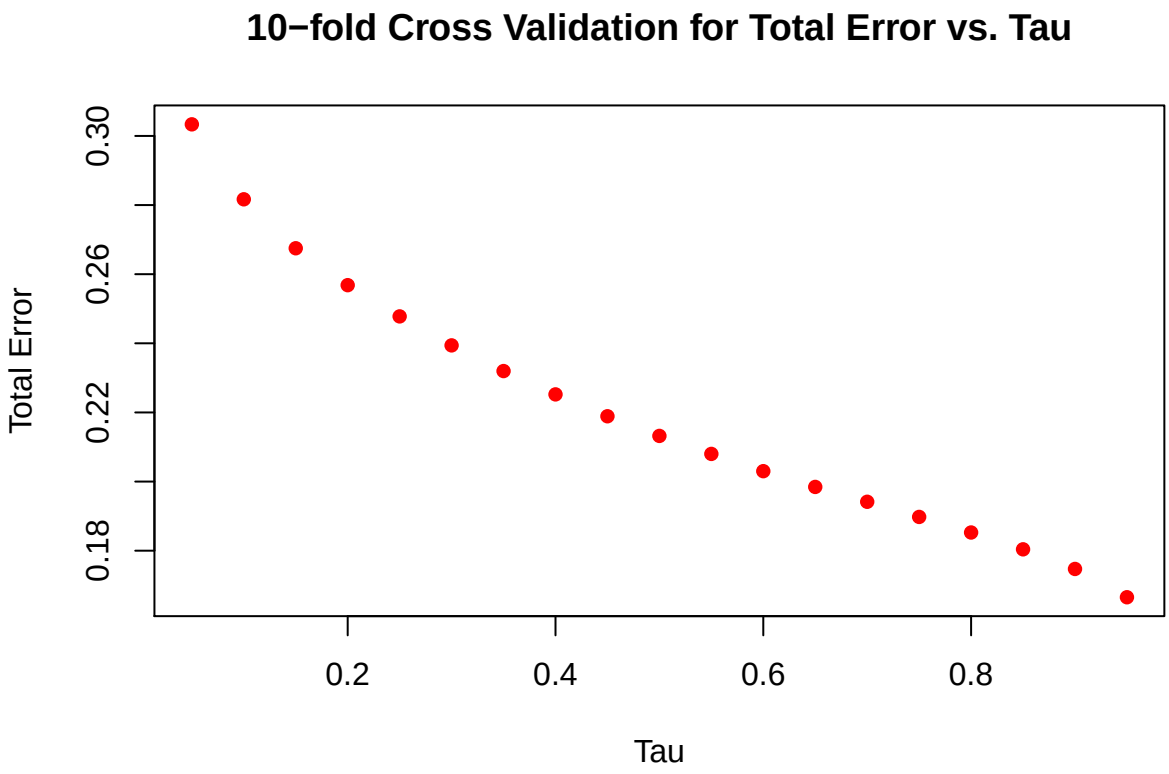
```
    return(data.frame(taus, spec_sens_calc, tot_err, avg_sens/k, avg_spec/k))
  }

  #Track average values
  avg_spec_qda <- mean(avg_spec)
  avg_sens_qda <- mean(avg_sens)
  avg_tot_err_qda <- mean(avg_tot_err)

  return (c(avg_spec_qda, avg_sens_qda, avg_tot_err_qda))
}
```

```
# Learn QDA model
out_qda <- calc_qda_vals(data = ghost_train_df, tau_val = 0.5, select_tau = TRUE)
```

```
# Plot 10-fold Cross Validation for Total Error vs. Tau - QDA
(out_qda %>% as_tibble() %>% arrange(tot_err) %>% head())
```

```
## # A tibble: 6 x 5
##    taus spec_sens_calc tot_err avg_sens.k avg_spec.k
##   <dbl>          <dbl>   <dbl>      <dbl>      <dbl>
## 1  0.95          0.809   0.167      0.771      0.846
## 2  0.9           0.811   0.175      0.788      0.833
## 3  0.85          0.811   0.180      0.799      0.824
## 4  0.8           0.812   0.185      0.807      0.816
## 5  0.75          0.811   0.190      0.813      0.810
## 6  0.7           0.811   0.194      0.819      0.803
```

```
plot(out_qda$taus, out_qda$tot_err, main = "10-fold Cross Validation for Total Error vs. Tau", ylab = "
```




```r
# Fit QDA for ghost particles
qda_ghost_fit <- MASS::qda(ghostLabel~.,data=ghost_train_df)
```

```r
# QDA with CV best tau
probs <- predict(qda_ghost_fit, ghost_val_df)$posterior[,2]
pred_ghost_cv <- ifelse(probs >= 0.95, 1, 0)
```

```r
u <- union(pred_ghost_cv, ghost_val_df$ghostLabel)
# Build backbone for confusion matrix
conf_matrix <- table(ordered(pred_ghost_cv, u, levels = c(0,1)), ordered(ghost_val_df$ghostLabel, u, le
#Generate a confusion matrix from the predicted and actual values
conf_mat <- caret::confusionMatrix(conf_matrix)$table
#Calculate sensitivity, specificity and total error
sens <- (conf_mat[2,2]/colSums(conf_mat)[2])[[1]]
spec <- (conf_mat[1,1]/colSums(conf_mat)[1])[[1]]
total_err <- (conf_mat[1,2] + conf_mat[2,1]) / sum(conf_mat)
qda_fnp <- (conf_mat[1,2]/rowSums(conf_mat)[1])[[1]]
qda_fpp <- (conf_mat[2, 1]/rowSums(conf_mat)[2])[[1]]
qda_sens <- sens
qda_spec <- spec
qda_err <- total_err
# Create a table of values for QDA
tbl_qda <- data.frame(qda_sens, qda_spec, qda_err, qda_fnp, qda_fpp)
```

```r
# Function to perform LDA and cross validation
calc_lda_vals <- function(data, select_tau = FALSE) {
  set.seed(1)
```

```r
#Set k folds
k <- 5

#Cut the data into k folds
folds <- cut(seq(1, nrow(data)), breaks = k, labels = FALSE)

# Keep track of average values for each fold
if (select_tau) {

  # Initialize Values
  taus <- seq(from = 0.05, to = 0.95, by = 0.05)
  avg_sens <- rep(0, times = length(taus))
  avg_spec <- rep(0, times = length(taus))
  avg_tot_err <- rep(0, times = length(taus))
} else {

  # Initialize Values
  avg_spec <- rep(0, times = k)
  avg_sens <- rep(0, times = k)
  avg_tot_err <- rep(0, times = k)
}

for(i in 1:k){

  #Segment data
    idx <- which(folds==i,arr.ind=TRUE)
    val <- data[idx, ]
    train <- data[-idx, ]


    #Fit the model
    fit <- MASS::lda(ghostLabel~.,data=train)

    #Find probabilities and predictions
    probs <- predict(fit, val)$posterior[,2]

    # Hyperparameter Search
    if (select_tau) {
      for(j in 1:length(taus)){

        # Update Hyperparameter vector
        param <- taus[j]
        pred <- ifelse(probs >= param, 1, 0)
        u <- union(pred, val$ghostLabel)
        conf_matrix <- table(ordered(pred, u, levels = c(0,1)),
                             ordered(val$ghostLabel, u, levels = c(0,1)))


        #Generate a confusion matrix from the predicted and actual values
        conf_mat <- caret::confusionMatrix(conf_matrix)$table

        #Calculate sensitivity, specificity and total error
        sens <- (conf_mat[2,2]/colSums(conf_mat)[2])[[1]]
```

```
          spec <- (conf_mat[1,1]/colSums(conf_mat)[1])[[1]]
          total_err <- (conf_mat[1,2] + conf_mat[2,1]) / sum(conf_mat)

          #Keep track of values for the current folds
          avg_spec[j] %+=% spec
          avg_sens[j] %+=% sens
          avg_tot_err[j] %+=% total_err
        }
      } else {

          # Generate class based upon probability
          pred <- ifelse(probs >= tau_val, 1, 0)

          #Generate a confusion matrix from the predicted and actual values
          u <- union(pred, val$y)
          conf_matrix <- table(ordered(pred, u, levels = c(0,1)),
                               ordered(val$y, u, levels = c(0,1)))
          conf_mat <- caret::confusionMatrix(conf_matrix)$table

          #Calculate sensitivity, specificity and total error
          sens <- (conf_mat[2,2]/colSums(conf_mat)[2])[[1]]
          spec <- (conf_mat[1,1]/colSums(conf_mat)[1])[[1]]
          total_err <- (conf_mat[1,2] + conf_mat[2,1]) / sum(conf_mat)

          #Keep track of values for the current folds
          avg_spec[i] = spec
          avg_sens[i] = sens
          avg_tot_err[i] = total_err
      }
  }
  if (select_tau) {

    # Fit metrics
    tot_err <- avg_tot_err/k
    spec_sens_calc <- (avg_spec/k + avg_sens/k )/ 2
    return(data.frame(taus, spec_sens_calc, tot_err, avg_sens/k, avg_spec/k))
  }

  #Track average values
  avg_spec_qda <- mean(avg_spec)
  avg_sens_qda <- mean(avg_sens)
  avg_tot_err_qda <- mean(avg_tot_err)
  return (c(avg_spec_qda, avg_sens_qda, avg_tot_err_qda))
}
```

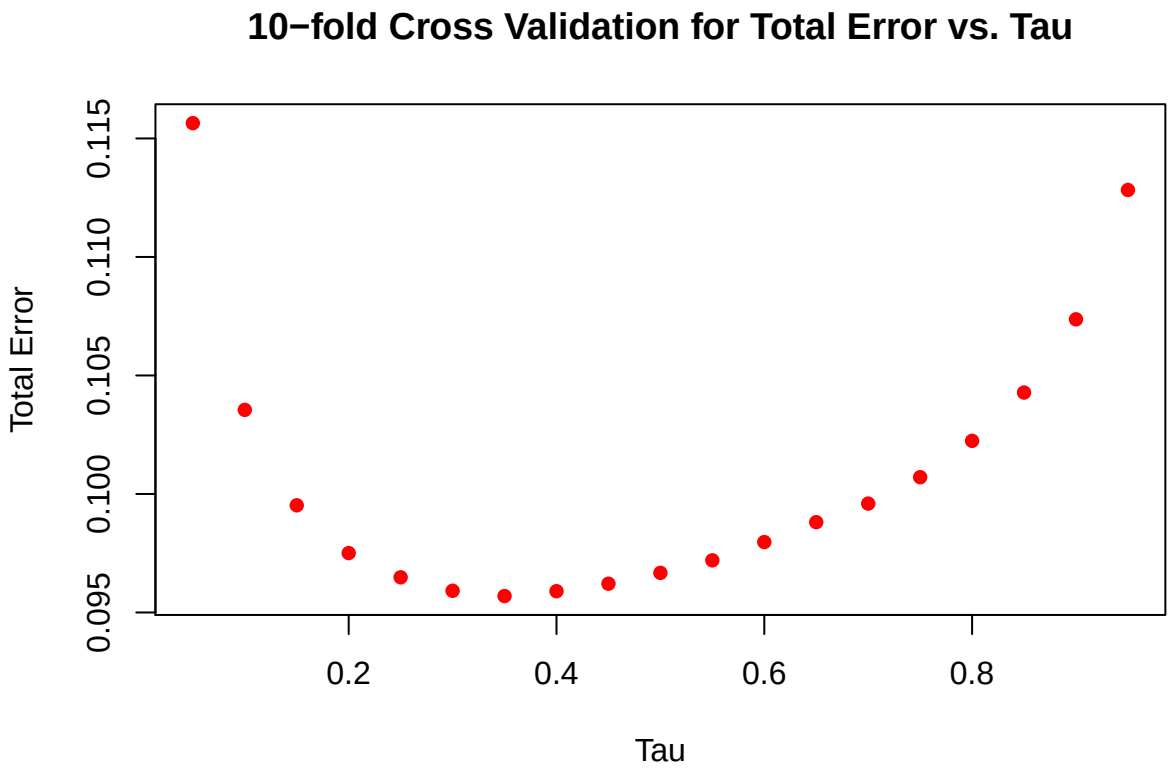
```
# Fit LDA models
out_lda <- calc_lda_vals(data = ghost_train_df, select_tau = TRUE)
```

```
# Plot 10-fold Cross Validation for Total Error vs. Tau - LDA
(out_lda %>% as_tibble() %>% arrange(tot_err) %>% head())
```

```
## # A tibble: 6 x 5
##    taus spec_sens_calc tot_err avg_sens.k avg_spec.k
##   <dbl>          <dbl>   <dbl>      <dbl>      <dbl>
```

```
## 1  0.35          0.811  0.0957      0.670      0.951
## 2  0.4           0.805  0.0959      0.655      0.954
## 3  0.3           0.817  0.0959      0.687      0.948
## 4  0.45          0.798  0.0962      0.640      0.957
## 5  0.25          0.824  0.0965      0.704      0.944
## 6  0.5           0.792  0.0967      0.626      0.959
```

```r
plot(out_lda$taus, out_lda$tot_err, main = "10-fold Cross Validation for Total Error vs. Tau", ylab = "
```




```r
# Fit LDA to predict ghosts
lda_ghost_fit <- MASS::lda(ghostLabel~.,data=ghost_train_df)
```

```r
# LDA with CV best tau
probs <- predict(lda_ghost_fit, ghost_val_df)$posterior[,2]
pred_ghost_cv_lda <- ifelse(probs >= 0.35, 1, 0)
```

```r
u <- union(pred_ghost_cv_lda, ghost_val_df$ghostLabel)
#Generate a confusion matrix from the predicted and actual values
conf_matrix <- table(ordered(pred_ghost_cv_lda, u, levels = c(0,1)), ordered(ghost_val_df$ghostLabel, u
conf_mat <- caret::confusionMatrix(conf_matrix)$table
#Calculate sensitivity, specificity and total error
sens <- (conf_mat[2,2]/colSums(conf_mat)[2])[[1]]
spec <- (conf_mat[1,1]/colSums(conf_mat)[1])[[1]]
total_err <- (conf_mat[1,2] + conf_mat[2,1]) / sum(conf_mat)
lda_fnp <- (conf_mat[1,2]/rowSums(conf_mat)[1])[[1]]
lda_fpp <- (conf_mat[2, 1]/rowSums(conf_mat)[2])[[1]]
lda_sens <- sens
```

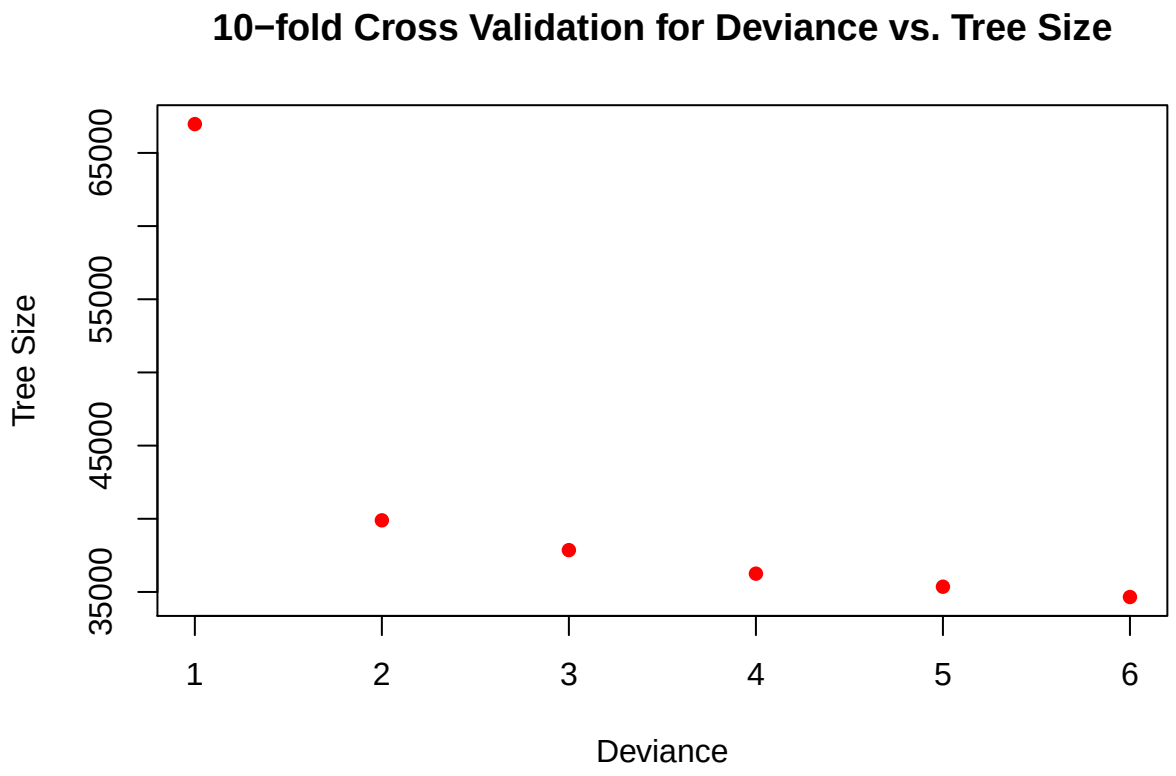
```
lda_spec <- spec
lda_err <- total_err
# Table of metrics for LDA
tbl_lda <- data.frame(lda_sens, lda_spec, lda_err, lda_fnp, lda_fpp)
tbl_lda
```

```
##    lda_sens  lda_spec    lda_err    lda_fnp   lda_fpp
## 1 0.6718203 0.9516968 0.09491667 0.06446699 0.2645963
```

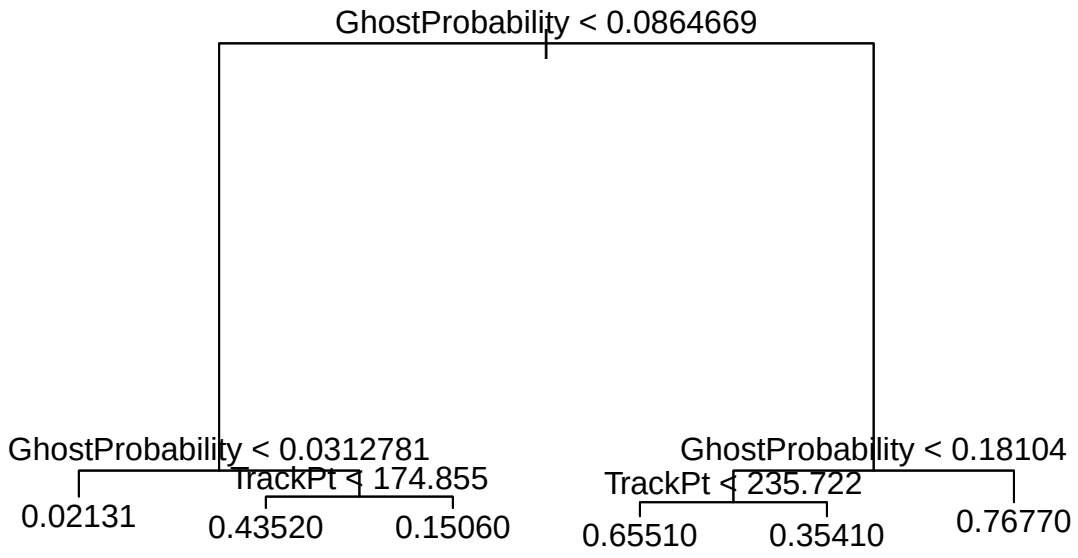
```
#Decision tree model
dec_tree_model <- tree::tree(ghostLabel ~ ., data = ghost_train_df )
```

```
set.seed(420)
# Cross validation for decision tree model
out_tree <- cv.tree(dec_tree_model ,FUN=prune.tree, K= 10)
```

```
# Plot 10-fold Cross Validation for Deviance vs. Tree Size
tree_df <- data.frame(out_tree$size, out_tree$dev)
plot(out_tree$size, out_tree$dev, main = "10-fold Cross Validation for Deviance vs. Tree Size", ylab =
```




```
# Prune Tree
pruned_model <- tree::prune.tree(dec_tree_model, best = 6)
# Visualize Tree
plot(pruned_model)
text(pruned_model ,pretty =0)
```




```
# Learn a decision tree to predict ghost or not
tree_pred <- predict(pruned_model ,ghost_val_df , type="vector")
pred_ghost_cv_tree <- ifelse(probs >= 0.5, 1, 0)
```

```
u <- union(pred_ghost_cv_tree, ghost_val_df$ghostLabel)
#Generate a confusion matrix from the predicted and actual values
conf_matrix <- table(ordered(pred_ghost_cv_tree, u, levels = c(0,1)), ordered(ghost_val_df$ghostLabel, u
conf_mat <- caret::confusionMatrix(conf_matrix)$table
#Calculate sensitivity, specificity and total error for Decision Tree
sens <- (conf_mat[2,2]/colSums(conf_mat)[2])[[1]]
spec <- (conf_mat[1,1]/colSums(conf_mat)[1])[[1]]
total_err <- (conf_mat[1,2] + conf_mat[2,1]) / sum(conf_mat)
tree_fnp <- (conf_mat[1,2]/rowSums(conf_mat)[1])[[1]]
tree_fpp <- (conf_mat[2, 1]/rowSums(conf_mat)[2])[[1]]
tree_sens <- sens
tree_spec <- spec
tree_err <- total_err
# Table of Metrics for decision tree
tbl_tree <- data.frame(tree_sens, tree_spec, tree_err, tree_fnp, tree_fpp)
tbl_tree
```

```
##   tree_sens tree_spec   tree_err   tree_fnp  tree_fpp
## 1 0.6283899 0.9598056 0.09539167 0.07181327 0.2424754
```

Analysis II

```
# Filter ghosts
noGhost <- data %>% dplyr::filter(Label != "Ghost")
noGhost$Label <- factor(noGhost$Label)
```

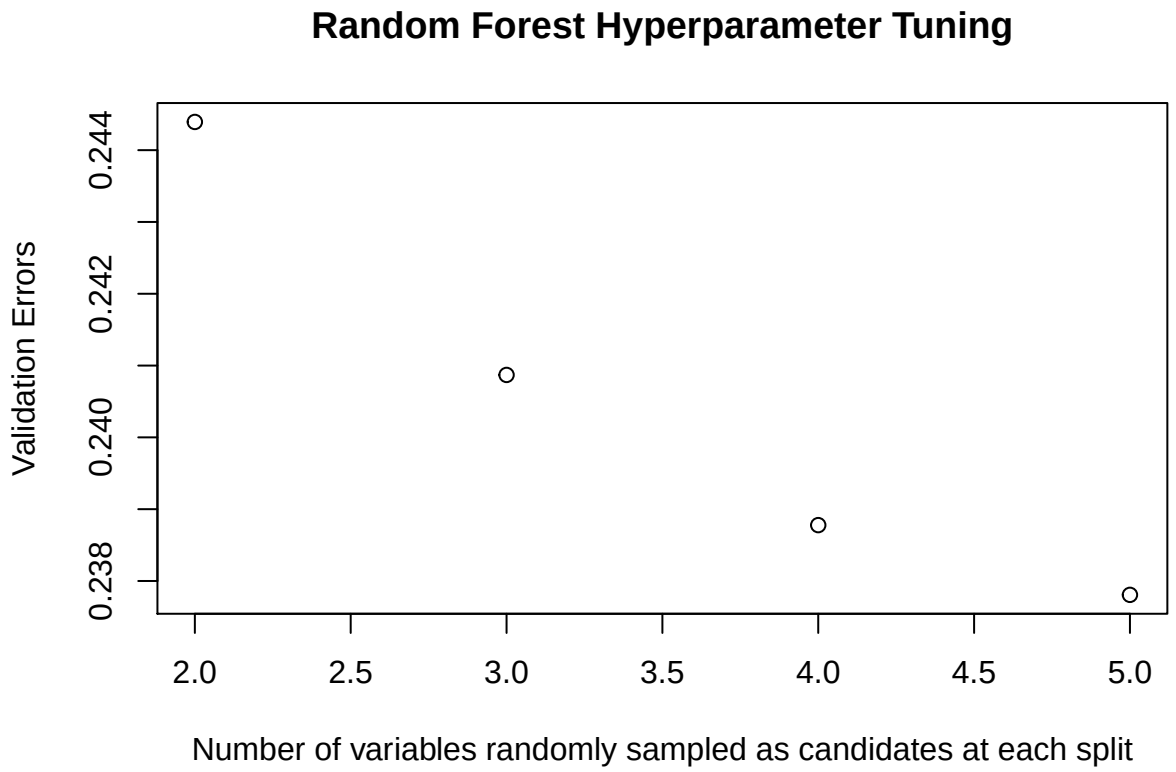
```
# Split no Ghost into train and validation sets
 set.seed(1)
#Find the sample size, which is 505 of data
smp_siz <- floor(0.8*nrow(noGhost))
#Randomly finds rows equal to sample size and indices of those rows
data_ind <- sample(seq_len(nrow(noGhost)),size = smp_siz)
#One dataset includes these randomly found rows, the other does not
train <- noGhost[data_ind,]
val <- noGhost[-data_ind,]
```

```
# Make Random Forest with Multiple values for mytry
 set.seed(3)
rf.2 <- randomForest(Label~. , data = train, mtry = 2, importance = T, ntree = 10)
rf.3 <- randomForest(Label~. , data = train, mtry = 3, importance = T, ntree = 10)
rf.4 <- randomForest(Label~. , data = train, mtry = 4, importance = T, ntree = 10)
rf.5 <- randomForest(Label~. , data = train, mtry = 5, importance = T, ntree = 10)
```

```
# RF 2 error
preds <- as.vector(predict(rf.2, type="class", newdata=val))
errors <- 0
for (i in 1:length(preds)){
  if(preds[i] != val$Label[i]){
    errors = errors + 1
  }
}
test.error.rf2 <- errors/length(preds)
# RF 3 error
preds <- as.vector(predict(rf.3, type="class", newdata=val))
errors <- 0
for (i in 1:length(preds)){
  if(preds[i] != val$Label[i]){
    errors = errors + 1
  }
}
test.error.rf3 <- errors/length(preds)
# RF 4 error
preds <- as.vector(predict(rf.4, type="class", newdata=val))
errors <- 0
for (i in 1:length(preds)){
  if(preds[i] != val$Label[i]){
    errors = errors + 1
  }
}
test.error.rf4 <- errors/length(preds)
# RF 5 error
preds <- as.vector(predict(rf.5, type="class", newdata=val))
errors <- 0
for (i in 1:length(preds)){
  if(preds[i] != val$Label[i]){
```

```
    errors = errors + 1
  }
}
test.error.rf5 <- errors/length(preds)
```

```
# Validation Erros Aggregated and Plotted
Validation.Errors <- c(test.error.rf2,
test.error.rf3,
test.error.rf4,
test.error.rf5)
mtry <- c(2 ,3, 4, 5)
plot(mtry, Validation.Errors, ylab = "Validation Errors", xlab = "Number of variables randomly sampled a
```




```
# Fit Random Forest using all the training data and the best hyperparameter
start.time <- Sys.time()
rf.4 <- randomForest(Label~. , data = noGhost, mtry = 4, importance = T, ntree = 10)
end.time <- Sys.time()
end.time - start.time
```

```
## Time difference of 2.582552 mins
```

```
# Fit a multiclass naive bayes model
start.time <- Sys.time()
nb.class <- e1071::naiveBayes(formula = Label~. ,data = train)
end.time <- Sys.time()
end.time - start.time
```

```
## Time difference of 1.165854 secs
```

```r
# Make predictions for naive bayes
preds <- predict(nb.class, type="class", newdata=val)
errors <- 0
for (i in 1:length(preds)){
  if(preds[i] != val$Label[i]){
    errors = errors + 1
  }
}
val.error.nb <- errors/length(preds)
val.error.nb
```

```
## [1] 0.5848488
```

```r
# Fit a QDA model
qdafit <- MASS::qda(Label~., data = train)
```

```r
# Predict with QDA
preds <- predict(qdafit, type="class", newdata=val)$class
errors <- 0
for (i in 1:length(preds)){
  if(preds[i] != val$Label[i]){
    errors = errors + 1
  }
}
test.error.qda <- errors/length(preds)
test.error.qda
```

```
## [1] 0.4847918
```

```r
# Fit a LDA model
ldafit <- MASS::lda(Label~., data = train)
```

```r
# Predict with LDA
preds <- predict(ldafit, type="class", newdata=val)$class
errors <- 0
for (i in 1:length(preds)){
  if(preds[i] != val$Label[i]){
    errors = errors + 1
  }
}
test.error.lda <- errors/length(preds)
test.error.lda
```

```
## [1] 0.2745489
```

```r
## Multinomial Logistic Regression
multi.log <- nnet::multinom(data = train, formula = Label~.)
```

```
## # weights:  255 (200 variable)
## initial  value 643210.252266
## iter  10 value 542605.204732
## iter  20 value 515276.180074
## iter  30 value 489177.135150
## iter  40 value 425111.124698
## iter  50 value 388881.088867
## iter  60 value 354233.059198
## iter  70 value 342944.600256
```

```
## iter  80 value 338290.185925
## iter  90 value 333215.931478
## iter 100 value 330763.496016
## final  value 330763.496016
## stopped after 100 iterations
```

```
# Multinomial predictions
preds <- predict(multi.log, type="class", newdata=val)
errors <- 0
for (i in 1:length(preds)){
  if(preds[i] != val$Label[i]){
    errors = errors + 1
  }
}
test.error.logistic <- errors/length(preds)
test.error.logistic
```

```
## [1] 0.3200284
```

Build Ensemble Model

```
# Read in data 1
data <- readr::read_csv("https://www.dropbox.com/s/34gwx6e5mwkts2u/data.csv?dl=1")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Label = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
# Train the Tree on the Final Model
data <- data %>% dplyr::select(-c(X1))
data %>% mutate(
  ghostLabel = ifelse(Label == "Ghost", 1, 0)
) -> data_withGhostColumn
data_withGhostColumn <- data_withGhostColumn %>% dplyr::select(-c("Label"))
```

```
# Load Testing data
data2 <- readr::read_csv("https://www.dropbox.com/s/34gwx6e5mwkts2u/DATA2.csv?dl=1")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Label = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
# Create dummy variable
data2 %>% mutate(
  ghostLabel = ifelse(Label == "Ghost", 1, 0)
) -> data2_withGhostColumn
data2_withGhostColumn <- data2_withGhostColumn %>% dplyr::select(-c("Label"))
```

```
# Fit and Prune final Tree
dec_tree_model <- tree::tree(ghostLabel ~ ., data = data_withGhostColumn )
```

```r
pruned_model <- tree::prune.tree(dec_tree_model, best = 6)
```

```r
# Remove index
data2 <- data2 %>% dplyr::select(-c(X1))
```

```r
# To predict with the tree, we must remove the class label
data2_noLabel <- data2 %>% dplyr::select(-c(Label))
```

```r
# For each observation in data 2, predict if we have ghost or not
preds_tree <- predict(pruned_model, type="vector", newdata=data2_noLabel)
```

```r
# Create class values from the probability values
preds_tree<-ifelse(preds_tree > 0.5, 1, 0)
```

```r
# Add prediction column
data2 %>% mutate(
  ghostPrediction = preds_tree
) -> data2_withGhostPredictions
```

```r
# Filter based on if a ghost particle was predicted or not
data2_withGhostPredictions %>% filter(ghostPrediction == 1) -> predictedGhostObservations
data2_withGhostPredictions %>% filter(ghostPrediction == 0) -> predictedNotGhostObservations
```

```r
# Format actual value into a 0-1 dummy variable
predictedGhostObservations %>% mutate(
  Actual = ifelse(Label == "Ghost", 1, 0)
) -> predictedGhostObservationReadyForComparison
```

```r
# Start calculating test error
labeledGhost <- nrow(predictedGhostObservationReadyForComparison)
errors.ghost <- 0
for (i in 1:nrow(predictedGhostObservationReadyForComparison)){
  if (predictedGhostObservationReadyForComparison$ghostPrediction[i] != predictedGhostObservationReadyF
    errors.ghost <- errors.ghost + 1
  }
}
```

```r
# Make predictions based on class using the random forest
rf.preds <- predictedNotGhostObservations
predictedNotGhostObservations$Label <- factor(predictedNotGhostObservations$Label)
preds <- as.vector(predict(rf.4, type="class", newdata=predictedNotGhostObservations))
# Count errors
errors.rf <- 0
for (i in 1:length(preds)){
  if(preds[i] != predictedNotGhostObservations$Label[i]){
    errors.rf = errors.rf + 1
  }
}
```

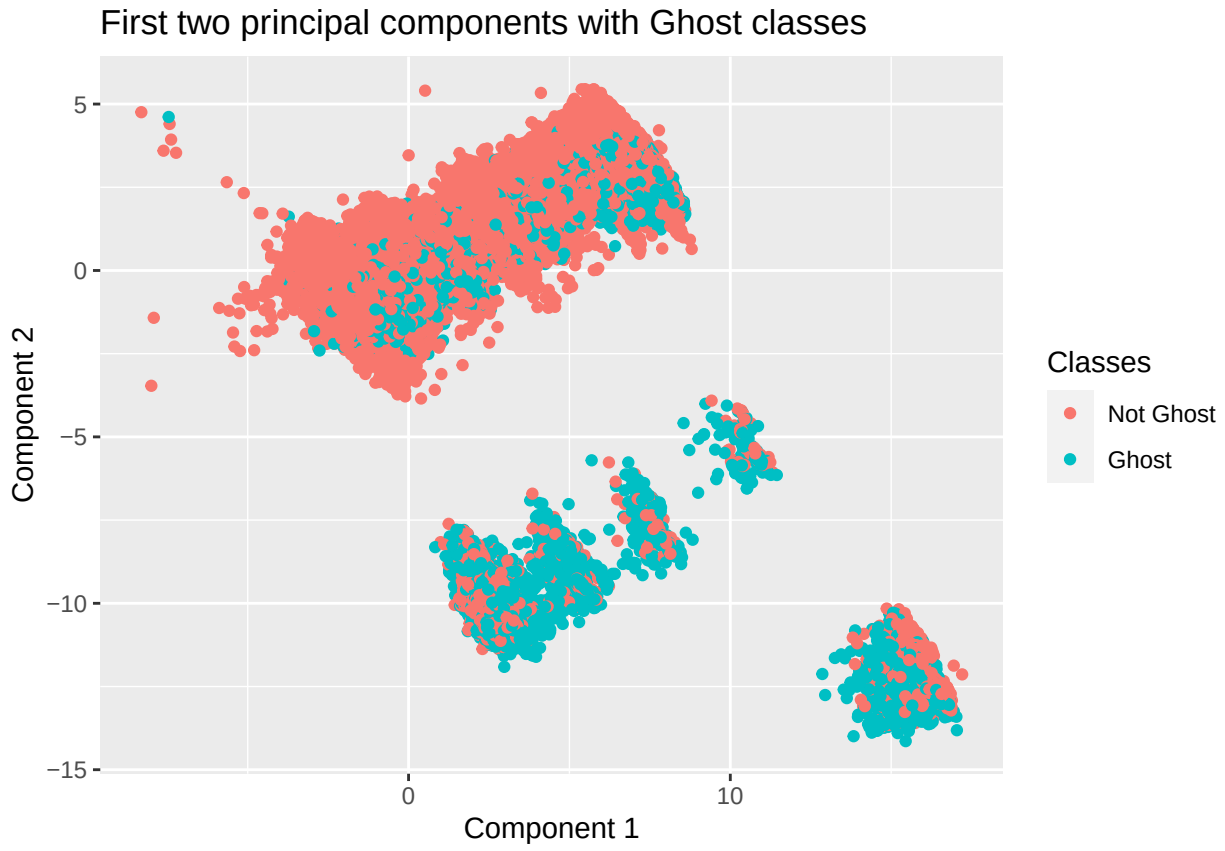
```r
# Calculate final error
total.error.final <- (errors.rf + errors.ghost)/(nrow(predictedNotGhostObservations) + labeledGhost)
total.error.final
```

```
## [1] 0.1098667
```

Principal Component Analysis

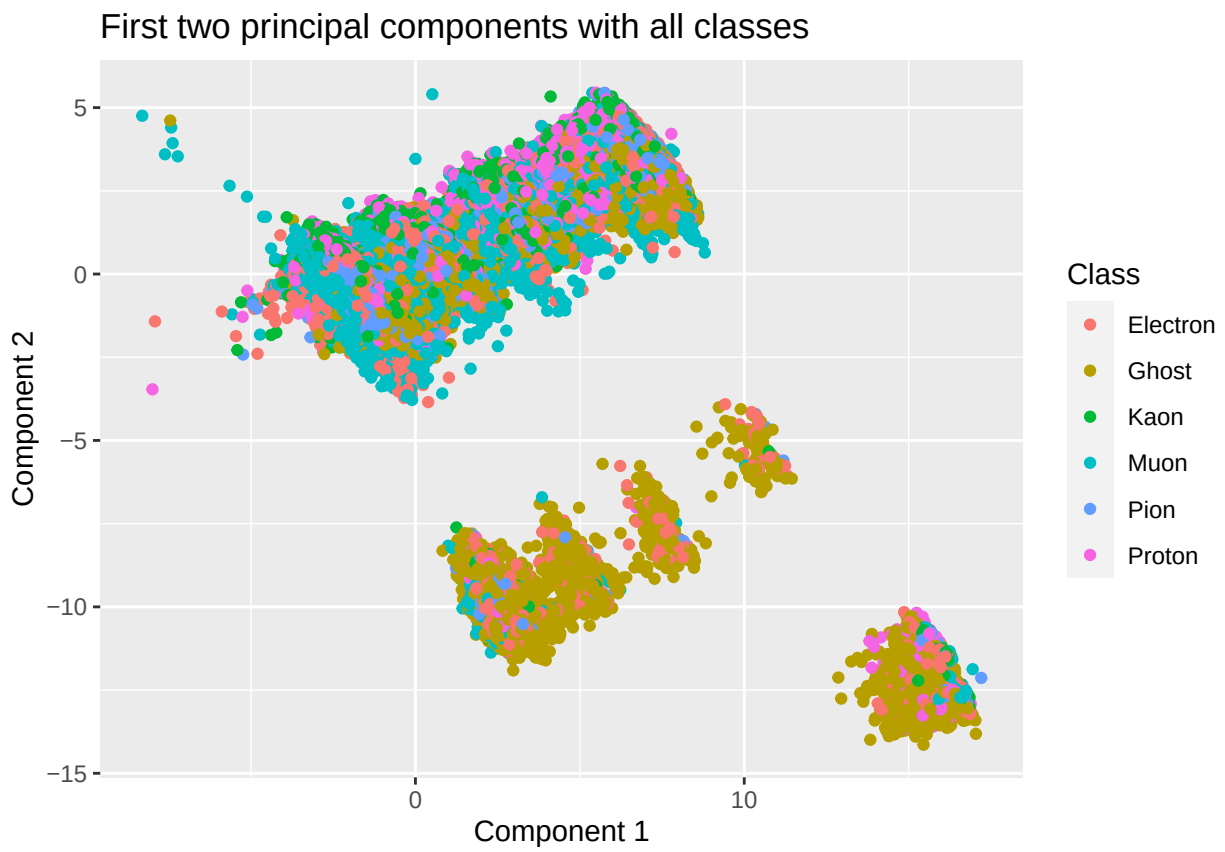
```
# PCA with ghosts
princomp_vals_ghost <- princomp(train_predictors
                                %>% scale(), cor = TRUE)
```

```
# Print PCA with ghost particles
pc_princomp_ghost <- princomp_vals_ghost$scores
pca_vals <- as.data.frame(pc_princomp_ghost)
# For legend purposes
pca_vals$class1 <- as.factor(ghost_train_labels)
# Plot PC1 vs PC 2 for ghost/not ghost
ggplot(data = pca_vals,
       aes(x = pca_vals$Comp.1, y = pca_vals$Comp.2, color= class1)) +
  geom_point() +
  scale_color_manual(name = "Classes", labels = c("Not Ghost", "Ghost"),
                     values = c("#F8766D", "#00BFC4")) +
  ggtitle("First two principal components with Ghost classes") +
  ylab("Component 2") + xlab("Component 1")
```



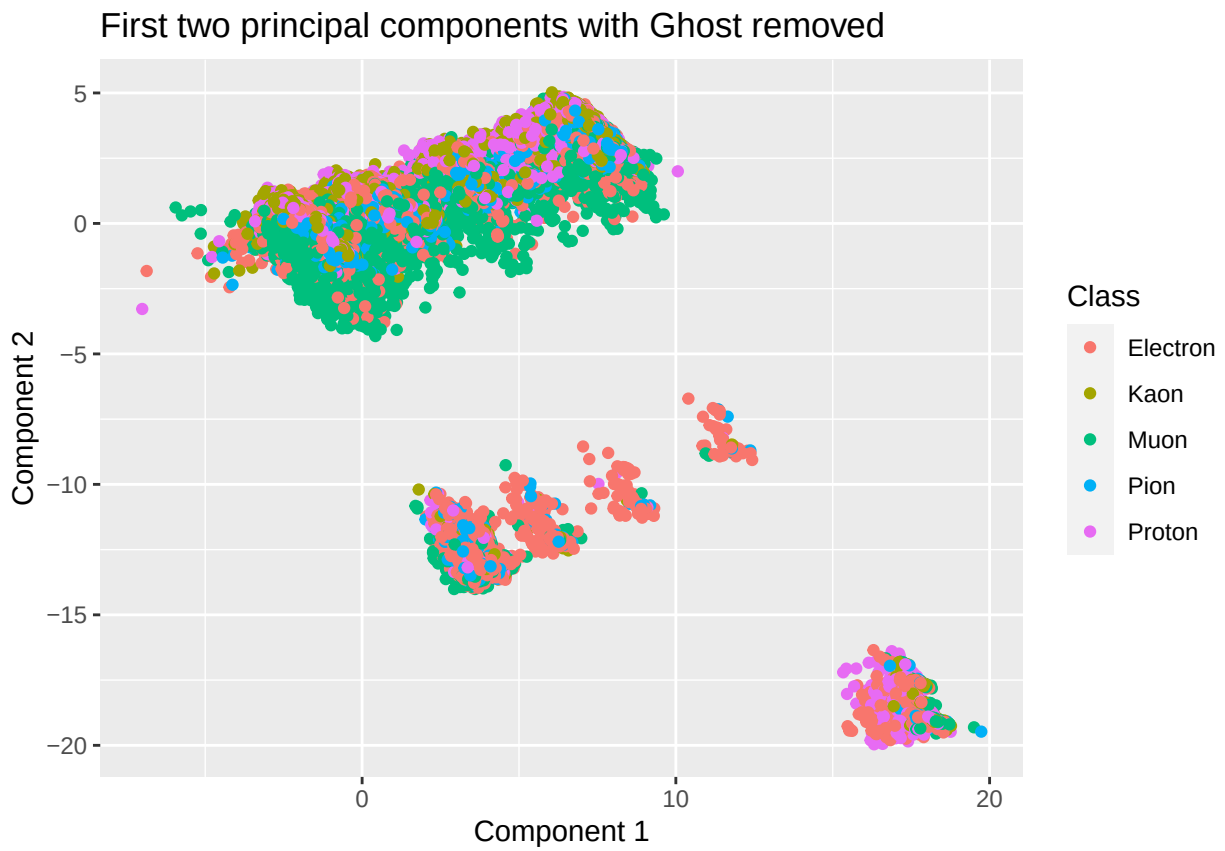

```
# PCA with all classes
pc_princomp_ghost <- princomp_vals_ghost$scores
pca_vals <- as.data.frame(pc_princomp_ghost)
pca_vals$Class <- as.factor(train_labels)
# Visualize
ggplot(data = pca_vals, aes(x = pca_vals$Comp.1, y = pca_vals$Comp.2, color= Class)) +
  geom_point()  +
```

```
  ggtitle("First two principal components with all classes") +
  ylab("Component 2") + xlab("Component 1")
```

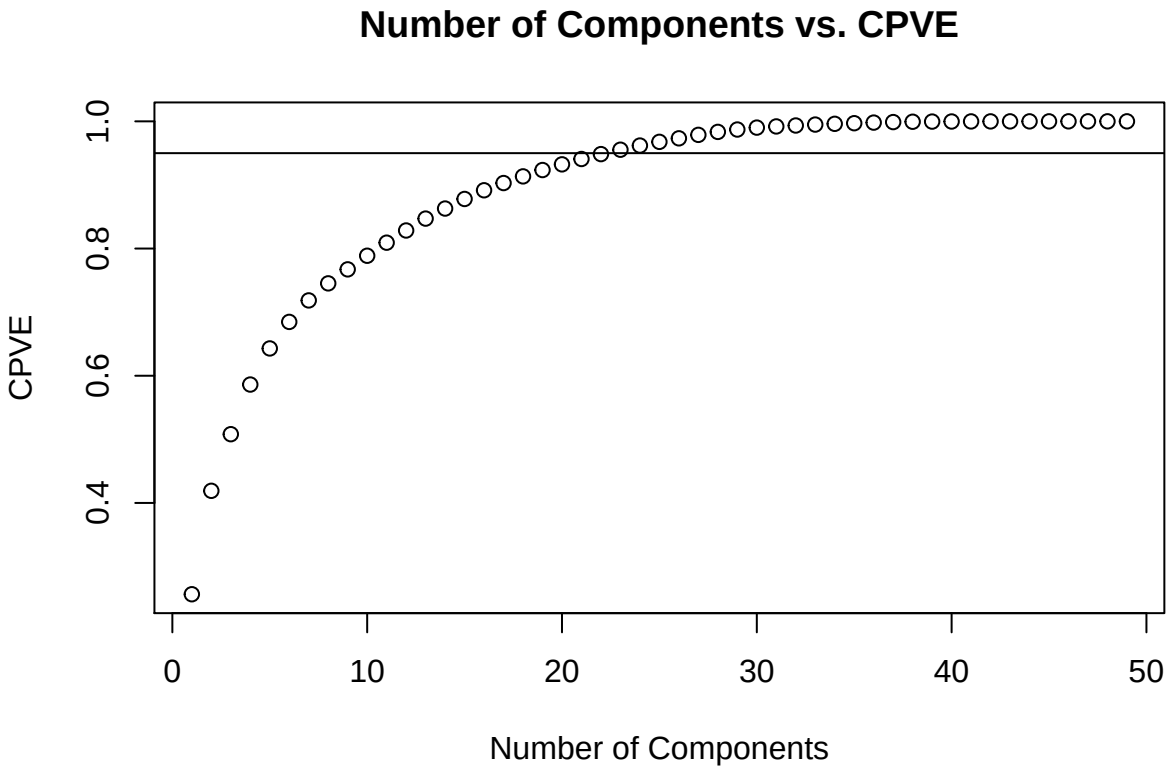



```
# PCA with ghost particles removed
princomp_vals_no_ghost <- princomp( no_ghost_train_predictors %>% scale(), cor = TRUE)
```

```
# Form data for visualization
pc_princomp_no_ghost <- princomp_vals_no_ghost$scores
pca_vals <- as.data.frame(pc_princomp_no_ghost)
pca_vals$Class <- as.factor(no_ghost_train_label)
# Visualize PCA with no ghosts
ggplot(data = pca_vals,
       aes(x = pca_vals$Comp.1, y =  -1 * pca_vals$Comp.2, color= Class)) +
  geom_point()  +
  ggtitle("First two principal components with Ghost removed") +
  ylab("Component 2") + xlab("Component 1")
```




```
# PCA from scratch
sig <- train_predictors %>% scale() %>% cov()
decomp <- eigen(sig)
lambda <- diag(decomp$values)
V <- decomp$vectors
sumLambda <- sum(decomp$values)
cumsums <- cumsum(decomp$values)
yvals <- sapply(cumsums, function(x){x/sumLambda})
# CPVE Plot
plot(1:49, yvals, main = "Number of Components vs. CPVE",
     xlab = "Number of Components", ylab = "CPVE")
abline(h = 0.95)
```


Number of Components vs. CPVE
CPVE
0.4
0.6
0.8
1.0
0
10
20
30
40
50
Number of Components