Saltmarsh Habitat Classification Models

Alyssa Bueno

2025-08-02

Saltmarsh Habitat Classification

This code outlines 4 different model classification iterations, each differing by the input layers.

1. NDWI + NDVI + PCA + NAIP + Brightness
2. NDWI + NDVI + PCA
3. NAIP + NDVI
4. PCA

Setup: Ingest the training data and set up the layers

```
# load raster and training polygons
naip <- rast("training_raster_round_6.tif") # this has the raster data
training_polygons <- vect("training_polygons_round_6.shp") # this has the classes

names(naip) <- paste0("naip", 1:4) # change name of naip bands layer

# calculate NDVI
ndvi <- (naip[[4]] - naip[[1]]) / (naip[[4]] + naip[[1]])
names(ndvi) <- "ndvi"

# calculate brightness
brightness <- (naip[[1]] + naip[[2]] + naip[[3]] + naip[[4]]) / 4
names(brightness) <- "brightness"

# calculate ndwi
ndwi <- (naip[[4]] - naip[[2]]) / (naip[[4]] + naip[[2]])
names(ndwi) <- "ndwi"

# now create the PCA

all_for_pca <- c(naip, ndvi)  # add ndvi to the raster
vals <- values(all_for_pca)
vals <- vals[complete.cases(vals), ]
pca_pixels <- prcomp(vals, center = TRUE, scale. = TRUE)
pca <- predict(all_for_pca, pca_pixels, index = 1:5)  # for 5 PCs
```

```
## |---------|---------|---------|---------|=========================================
```

```r
names(pca) <- paste0("PCA", 1:5)
```

1. First Iteration: NDWI + NDVI + PCA + NAIP + Brightness

```r
# stack em up
r_stack <- c(ndwi, ndvi, pca, naip, brightness)

# extract raster values for training polygons
extracted <- terra::extract(r_stack, training_polygons, df = TRUE)

# turn class into a factor
training_polygons$class <- as.factor(training_polygons$class)

# Convert training_polygons to dataframe to get the class labels
poly_df <- as.data.frame(training_polygons)
poly_df$ID <- 1:nrow(poly_df)  # Add ID column to match extract output

# Join the class labels
extracted <- extracted %>%
  left_join(poly_df[, c("ID", "class")], by = "ID")

# clean data by removing rows with na values and remove ID column
extracted_clean <- na.omit(extracted[, -1])  # remove ID and NAs

# sample 2000 per class
training_data <- extracted_clean %>%
  group_by(class) %>%
  sample_n(min(2000, n())) %>%  # Use min() to handle small classes
  ungroup()

# turn class column into factor
training_data$class <- factor(training_data$class)

# check the sampling balance in the classes
print(table(training_data$class))
```

```
##
##   hm   lm   md   ow   ph   rd   up
## 2000 2000 2000 2000 2000 2000 2000
```

```r
# split training and test data
set.seed(342)
idx <- sample(seq_len(nrow(training_data)), size = 0.8 * nrow(training_data))
train_set <- training_data[idx, ]
test_set  <- training_data[-idx, ]

# train random forest model
rf_model <- randomForest(class ~ .,
                         data = train_set,
                         ntree = 500)
```

```r
# validation metrics
preds <- predict(rf_model, newdata = test_set)
accuracy <- mean(preds == test_set$class)
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.9721
```

```r
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = class ~ ., data = train_set, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 2.12%
## Confusion matrix:
##      hm   lm   md   ow   ph   rd   up class.error
## hm 1596    8    0    0    1    2    0 0.006845053
## lm   13 1554    5    0   24    0    1 0.026925485
## md    0    2 1563    0    0   36    0 0.023735166
## ow    0    0    0 1589    0    0    0 0.000000000
## ph    0   34    0    0 1564    2   12 0.029776675
## rd    2    0   47    0   12 1543    1 0.038629283
## up    3   16    0    0   16    0 1554 0.022026432
```

```r
# confusion matrix
confusionMatrix(preds, test_set$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  hm  lm  md  ow  ph  rd  up
##         hm 388   4   0   0   0   1   0
##         lm   4 390   2   0  10   1   6
##         md   0   0 383   0   0  15   0
##         ow   0   0   0 411   0   0   0
##         ph   0   9   0   0 377   2   8
##         rd   1   0  14   0   0 376   0
##         up   0   0   0   0   1   0 397
##
## Overall Statistics
##
##                Accuracy : 0.9721
##                  95% CI : (0.9654, 0.9779)
##     No Information Rate : 0.1468
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9675
```

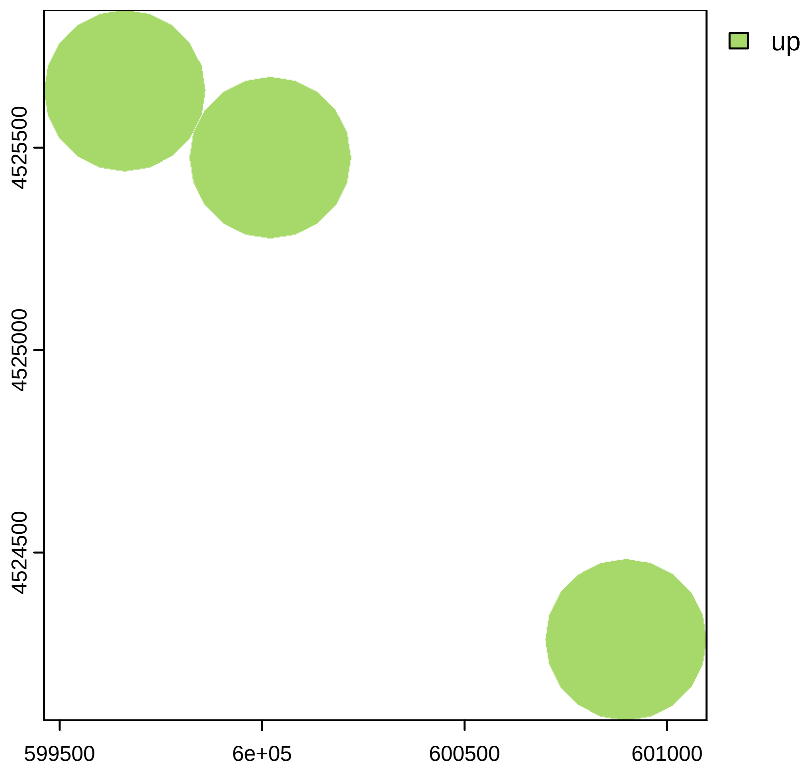
```
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: hm Class: lm Class: md Class: ow Class: ph
## Sensitivity             0.9873    0.9677    0.9599    1.0000    0.9716
## Specificity             0.9979    0.9904    0.9938    1.0000    0.9921
## Pos Pred Value          0.9873    0.9443    0.9623    1.0000    0.9520
## Neg Pred Value          0.9979    0.9946    0.9933    1.0000    0.9954
## Prevalence              0.1404    0.1439    0.1425    0.1468    0.1386
## Detection Rate          0.1386    0.1393    0.1368    0.1468    0.1346
## Detection Prevalence    0.1404    0.1475    0.1421    0.1468    0.1414
## Balanced Accuracy       0.9926    0.9791    0.9768    1.0000    0.9819
##                      Class: rd Class: up
## Sensitivity             0.9519    0.9659
## Specificity             0.9938    0.9996
## Pos Pred Value          0.9616    0.9975
## Neg Pred Value          0.9921    0.9942
## Prevalence              0.1411    0.1468
## Detection Rate          0.1343    0.1418
## Detection Prevalence    0.1396    0.1421
## Balanced Accuracy       0.9728    0.9828
```

classifying the plots

```r
# load the new, unlabeled raster
prediction_raster <-
↪  rast("~/Desktop/marshbirdsoutput/round_6/prediction_raster_round_6.tif")

# calculate NDVI for prediction raster
ndvi_pred <- (prediction_raster[[4]] - prediction_raster[[1]]) /
  (prediction_raster[[4]] + prediction_raster[[1]])
names(ndvi_pred) <- "ndvi"

# calculation NDWI
ndwi_pred <- (prediction_raster[[4]] - prediction_raster[[2]]) /
  (prediction_raster[[4]] + prediction_raster[[2]])
names(ndwi_pred) <- "ndwi"

# Calculate brightness
brightness_pred <- (prediction_raster[[1]] + prediction_raster[[2]] +
                      prediction_raster[[3]] + prediction_raster[[4]]) / 4
names(brightness_pred) <- "brightness"

# Apply the pca
all_for_pca_pred <- c(prediction_raster, ndvi_pred)
names(all_for_pca_pred) <- names(all_for_pca)  # ensure exact match
pca_pred <- predict(all_for_pca_pred, pca_pixels, index = 1:5)
```

```
## |---------|---------|---------|---------|=========================================
```

```r
names(pca_pred) <- paste0("PCA", 1:5)

# stack it
prediction_stack <- c(prediction_raster, ndvi_pred, ndwi_pred, brightness_pred, pca_pred)

# rename to match training names
names(prediction_stack) <- names(r_stack)

# apply the model to predict classes
classified <- predict(prediction_stack, rf_model, na.rm = TRUE)

# Define custom colors
colors <- c(
  "#a6d96a",  # hm
  "#1a9641",  # lm
  "#8c510a",  # md
  "#3288bd",  # ow
  "#fdae61",  # ph
  "#969696",  # rd
  "#762a83"   # up
)



# Plot with colors
plot(classified, col = colors)
```

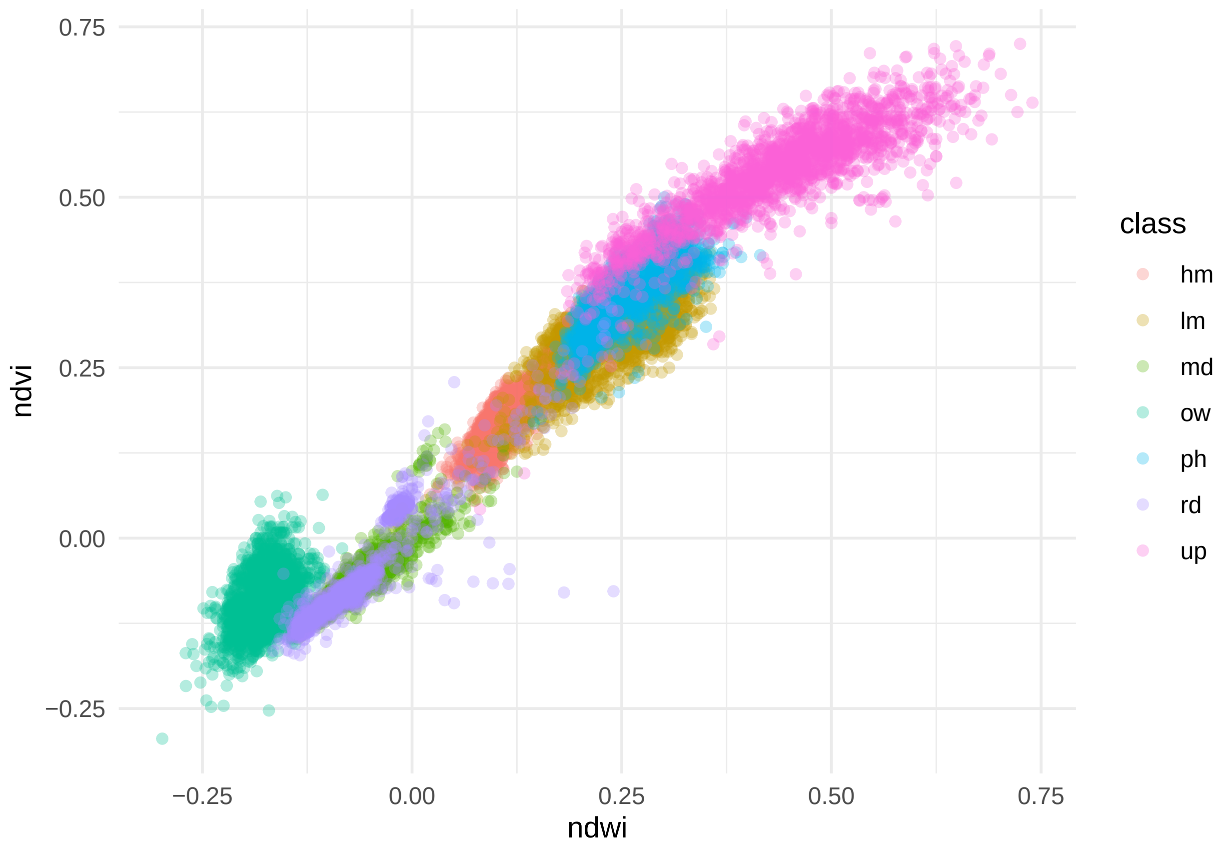



```r
# save the output tif
# writeRaster(classified, "~/Desktop/marshbirdsoutput/round_6/classified_output.tif",
↪  overwrite = TRUE)

rf_model$importance
```

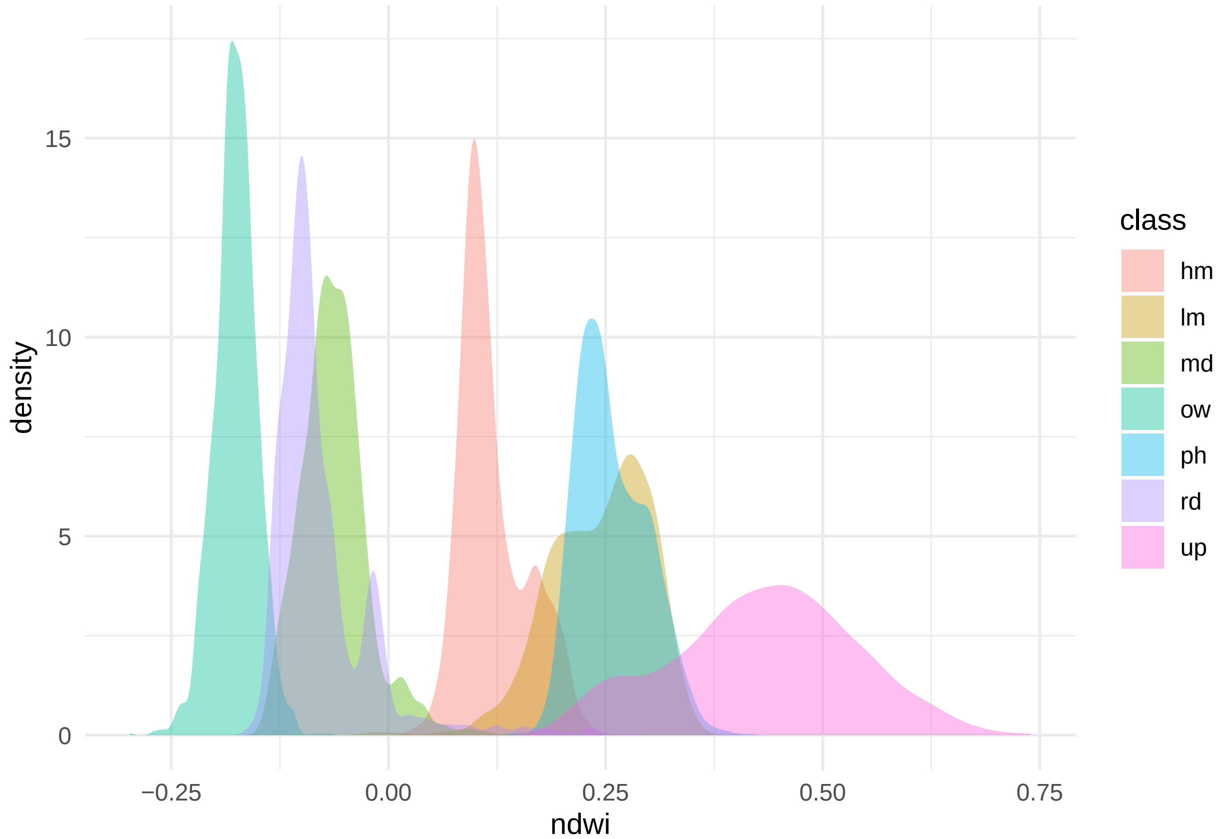
```
##            MeanDecreaseGini
## ndwi              1342.0737
## ndvi              1257.4674
## PCA1               802.3560
## PCA2               750.9137
## PCA3               540.1606
## PCA4               296.1795
## PCA5               387.6126
## naip1              777.2933
## naip2              482.3840
## naip3              904.0095
## naip4             1230.8604
## brightness         826.3936
```

feature class importance and visualizations

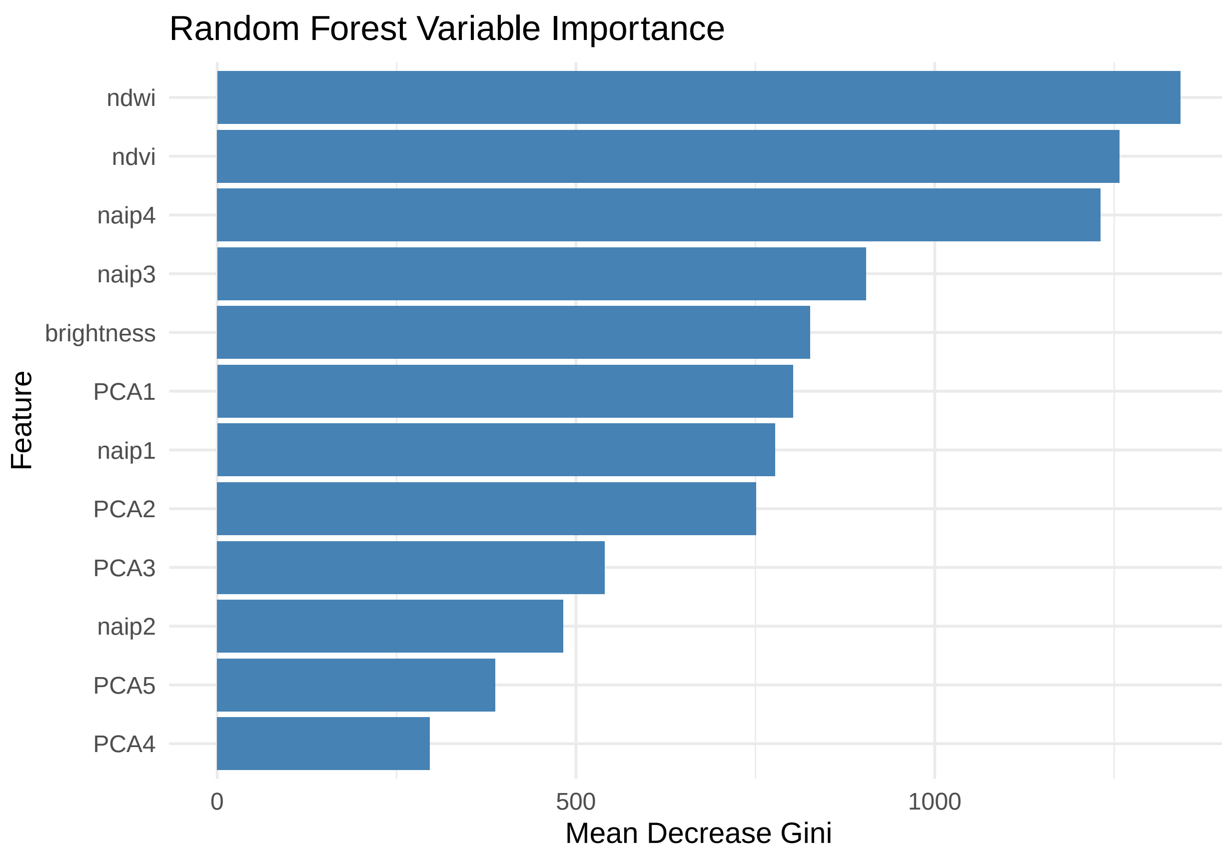
```r
ggplot(training_data, aes(x = ndwi, y = ndvi, color = class)) +
  geom_point(alpha = 0.3) +
  theme_minimal()
```




```r
ggplot(training_data, aes(x = ndwi, fill = class)) +
  geom_density(alpha = 0.4, color = NA) +
  theme_minimal()
```




```r
imp <- as.data.frame(rf_model$importance)
imp$feature <- rownames(imp)

ggplot(imp, aes(x = reorder(feature, MeanDecreaseGini), y = MeanDecreaseGini)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(title = "Random Forest Variable Importance",
       x = "Feature", y = "Mean Decrease Gini") +
  theme_minimal()
```



2. Second Iteration: NAIP + NDVI + NDWI

```
# stack em up
r_stack <- c(naip, ndvi, ndwi)

# extract raster values for training polygons
extracted <- terra::extract(r_stack, training_polygons, df = TRUE)

# turn class into a factor
training_polygons$class <- as.factor(training_polygons$class)

# Convert training_polygons to dataframe to get the class labels
poly_df <- as.data.frame(training_polygons)
poly_df$ID <- 1:nrow(poly_df)  # Add ID column to match extract output

# Join the class labels
extracted <- extracted %>%
  left_join(poly_df[, c("ID", "class")], by = "ID")

# clean data by removing rows with na values and remove ID column
extracted_clean <- na.omit(extracted[, -1])  # remove ID and NAs

# sample 2000 per class
```

```r
training_data <- extracted_clean %>%
  group_by(class) %>%
  sample_n(min(2000, n())) %>%  # Use min() to handle small classes
  ungroup()

# turn class column into factor
training_data$class <- factor(training_data$class)
```

```r
# split training and test data
set.seed(342)
idx <- sample(seq_len(nrow(training_data)), size = 0.8 * nrow(training_data))
train_set <- training_data[idx, ]
test_set  <- training_data[-idx, ]

# train random forest model
rf_model <- randomForest(class ~ .,
                         data = train_set,
                         ntree = 500)

# validation metrics
preds <- predict(rf_model, newdata = test_set)
accuracy <- mean(preds == test_set$class)
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.9696
```

```r
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = class ~ ., data = train_set, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 2.68%
## Confusion matrix:
##      hm   lm   md   ow   ph   rd   up  class.error
## hm 1595    8    0    0    0    3    1 0.0074673304
## lm   17 1544    3    0   31    2    0 0.0331872260
## md    0    2 1549    0    0   50    0 0.0324797002
## ow    0    0    1 1588    0    0    0 0.0006293266
## ph    0   47    0    0 1552    0   13 0.0372208437
## rd    9    4   53    0   16 1521    2 0.0523364486
## up    3   16    0    0   19    0 1551 0.0239144116
```

```r
# confusion matrix
confusionMatrix(preds, test_set$class)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  hm  lm  md  ow  ph  rd  up
##         hm 391   4   0   0   0   1   2
##         lm   2 389   0   0  17   0   3
##         md   0   1 385   0   0  19   0
##         ow   0   0   0 411   0   0   0
##         ph   0   9   0   0 368   4   6
##         rd   0   0  14   0   0 371   0
##         up   0   0   0   0   3   0 400
##
## Overall Statistics
##
##                Accuracy : 0.9696
##                  95% CI : (0.9626, 0.9757)
##     No Information Rate : 0.1468
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9646
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: hm Class: lm Class: md Class: ow Class: ph
## Sensitivity             0.9949    0.9653    0.9649    1.0000    0.9485
## Specificity             0.9971    0.9908    0.9917    1.0000    0.9921
## Pos Pred Value          0.9824    0.9465    0.9506    1.0000    0.9509
## Neg Pred Value          0.9992    0.9941    0.9942    1.0000    0.9917
## Prevalence              0.1404    0.1439    0.1425    0.1468    0.1386
## Detection Rate          0.1396    0.1389    0.1375    0.1468    0.1314
## Detection Prevalence    0.1421    0.1468    0.1446    0.1468    0.1382
## Balanced Accuracy       0.9960    0.9780    0.9783    1.0000    0.9703
##                      Class: rd Class: up
## Sensitivity             0.9392    0.9732
## Specificity             0.9942    0.9987
## Pos Pred Value          0.9636    0.9926
## Neg Pred Value          0.9901    0.9954
## Prevalence              0.1411    0.1468
## Detection Rate          0.1325    0.1429
## Detection Prevalence    0.1375    0.1439
## Balanced Accuracy       0.9667    0.9860
```

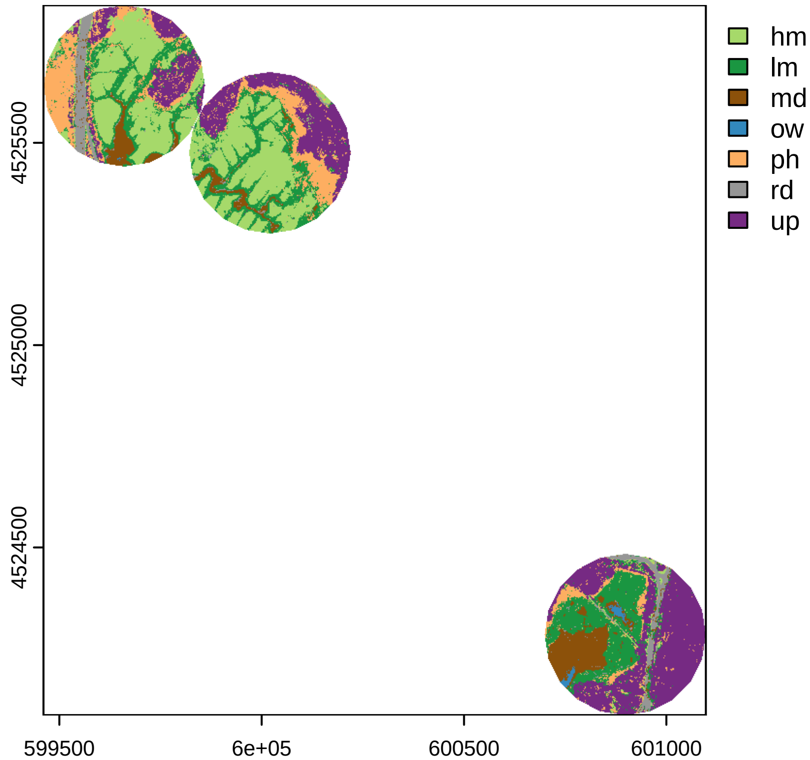
classifying the plots

```r
# load the new, unlabeled raster
prediction_raster <-
↪  rast("~/Desktop/marshbirdsoutput/round_6/prediction_raster_round_6.tif")

# calculate NDVI for prediction raster
ndvi_pred <- (prediction_raster[[4]] - prediction_raster[[1]]) /
  (prediction_raster[[4]] + prediction_raster[[1]])
```

```r
names(ndvi_pred) <- "ndvi"

# calculation NDWI
ndwi_pred <- (prediction_raster[[4]] - prediction_raster[[2]]) /
  (prediction_raster[[4]] + prediction_raster[[2]])
names(ndwi_pred) <- "ndwi"

# Calculate brightness
brightness_pred <- (prediction_raster[[1]] + prediction_raster[[2]] +
                      prediction_raster[[3]] + prediction_raster[[4]]) / 4
names(brightness_pred) <- "brightness"

# Apply the pca
all_for_pca_pred <- c(prediction_raster, ndvi_pred)
names(all_for_pca_pred) <- names(all_for_pca)  # ensure exact match
pca_pred <- predict(all_for_pca_pred, pca_pixels, index = 1:5)
```

```
## |---------|---------|---------|---------|=========================================
```

```r
names(pca_pred) <- paste0("PCA", 1:5)

# stack it
prediction_stack <- c(prediction_raster, ndvi_pred, ndwi_pred)

# rename to match training names
names(prediction_stack) <- names(r_stack)

# apply the model to predict classes
classified <- predict(prediction_stack, rf_model, na.rm = TRUE)

# Define custom colors
colors <- c(
  "#a6d96a",  # hm
  "#1a9641",  # lm
  "#8c510a",  # md
  "#3288bd",  # ow
  "#fdae61",  # ph
  "#969696",  # rd
  "#762a83"   # up
)



# Plot with colors
plot(classified, col = colors)
```

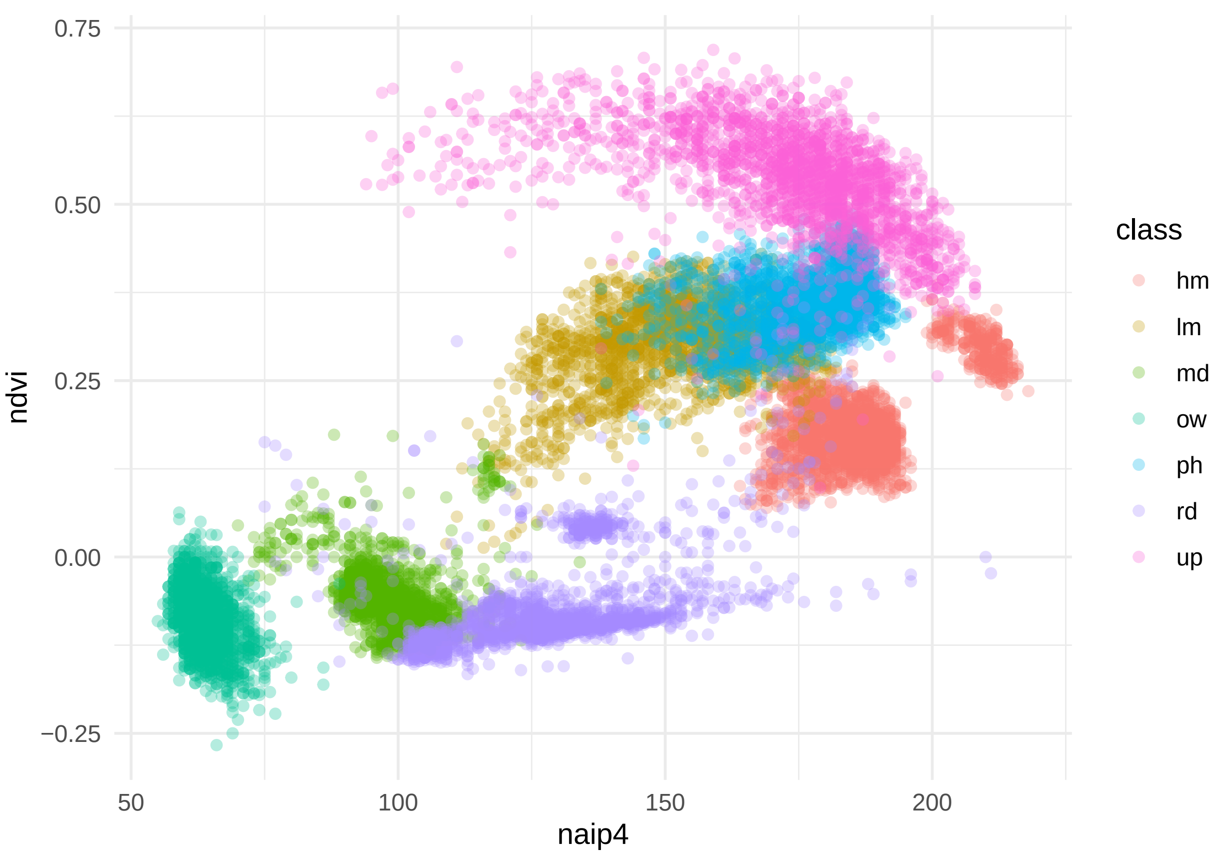



```r
# save the output tif
# writeRaster(classified, "~/Desktop/marshbirdsoutput/round_6/classified_output.tif",
↪  overwrite = TRUE)

model <- rf_model$importance
print(model)
```

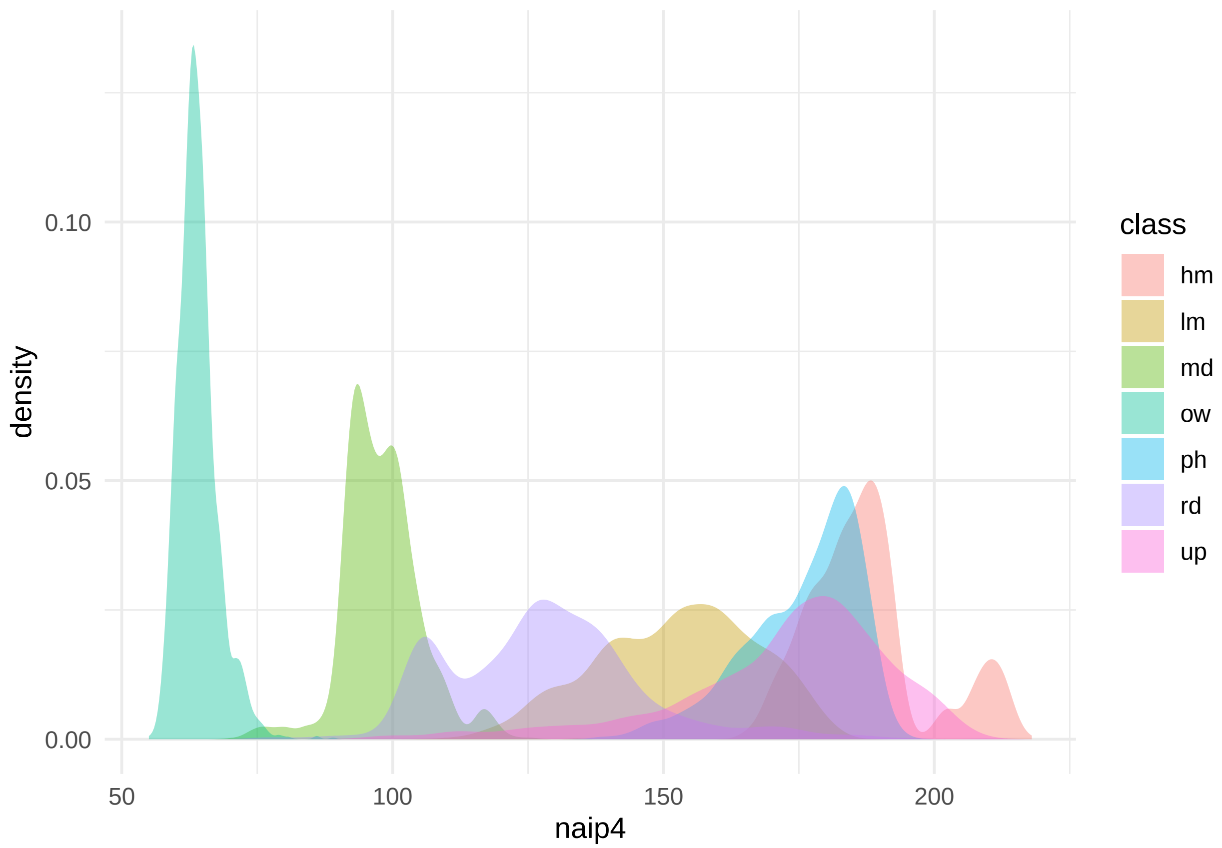
```
##       MeanDecreaseGini
## naip1         1287.991
## naip2         1177.615
## naip3         1668.540
## naip4         1862.514
## ndvi          1680.766
## ndwi          1915.391
```

feature class importance and visualizations

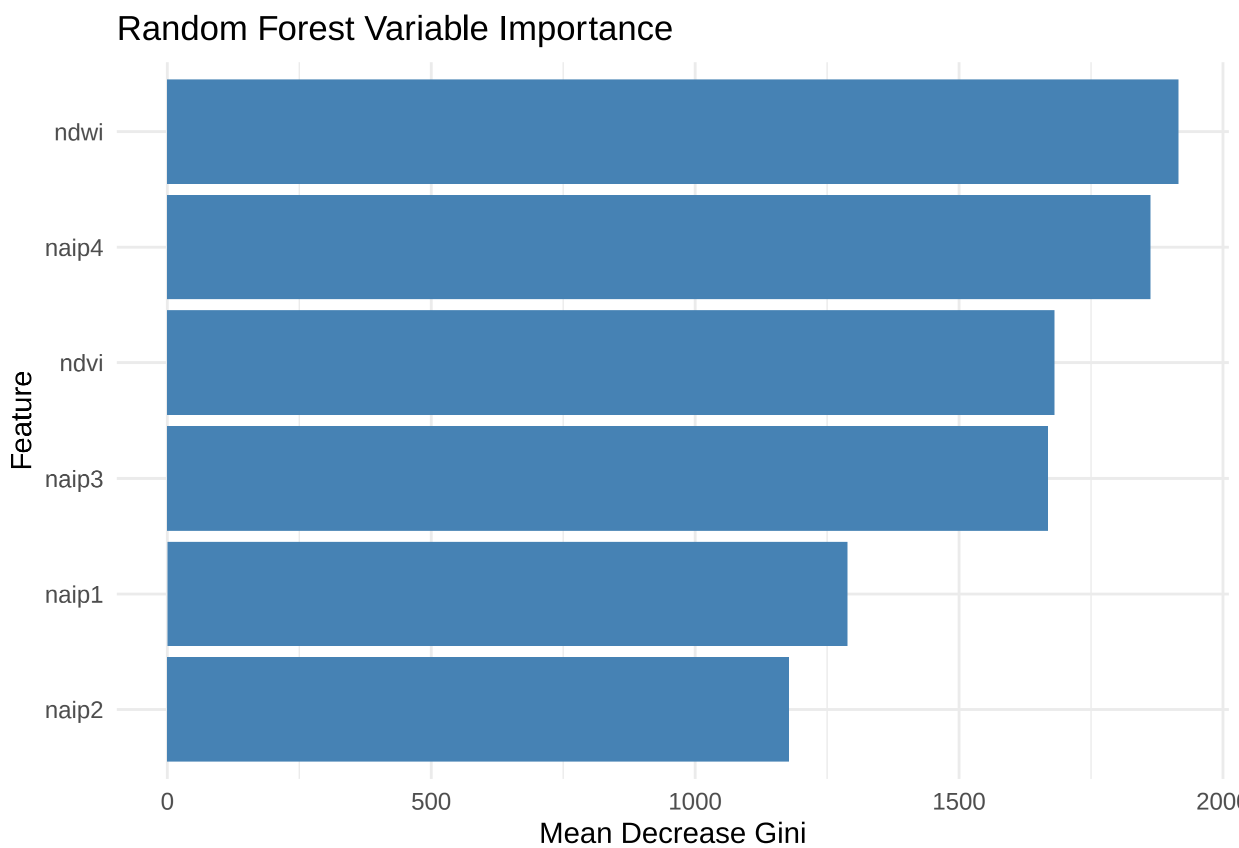
```r
ggplot(training_data, aes(x = naip4, y = ndvi, color = class)) +
  geom_point(alpha = 0.3) +
  theme_minimal()
```




```
ggplot(training_data, aes(x = naip4, fill = class)) +
  geom_density(alpha = 0.4, color = NA) +
  theme_minimal()
```




```r
imp <- as.data.frame(rf_model$importance)
imp$feature <- rownames(imp)

ggplot(imp, aes(x = reorder(feature, MeanDecreaseGini), y = MeanDecreaseGini)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(title = "Random Forest Variable Importance",
       x = "Feature", y = "Mean Decrease Gini") +
  theme_minimal()
```



3. Third Iteration: NAIP + NDVI

```
# stack em up
r_stack <- c(naip, ndvi)

# extract raster values for training polygons
extracted <- terra::extract(r_stack, training_polygons, df = TRUE)

# turn class into a factor
training_polygons$class <- as.factor(training_polygons$class)

# Convert training_polygons to dataframe to get the class labels
poly_df <- as.data.frame(training_polygons)
poly_df$ID <- 1:nrow(poly_df)  # Add ID column to match extract output

# Join the class labels
extracted <- extracted %>%
  left_join(poly_df[, c("ID", "class")], by = "ID")

# clean data by removing rows with na values and remove ID column
extracted_clean <- na.omit(extracted[, -1])  # remove ID and NAs

# sample 2000 per class
```

```r
training_data <- extracted_clean %>%
  group_by(class) %>%
  sample_n(min(2000, n())) %>%  # Use min() to handle small classes
  ungroup()

# turn class column into factor
training_data$class <- factor(training_data$class)
```

```r
# split training and test data
set.seed(342)
idx <- sample(seq_len(nrow(training_data)), size = 0.8 * nrow(training_data))
train_set <- training_data[idx, ]
test_set  <- training_data[-idx, ]

# train random forest model
rf_model <- randomForest(class ~ .,
                         data = train_set,
                         ntree = 500)

# validation metrics
preds <- predict(rf_model, newdata = test_set)
accuracy <- mean(preds == test_set$class)
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.9686
```

```r
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = class ~ ., data = train_set, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 2.97%
## Confusion matrix:
##      hm   lm   md   ow   ph   rd   up class.error
## hm 1592   11    0    0    0    4    0 0.009334163
## lm   20 1534    3    0   39    0    1 0.039448967
## md    0    1 1551    0    0   49    0 0.031230481
## ow    0    0    1 1587    0    1    0 0.001258653
## ph    0   58    0    0 1535    2   17 0.047766749
## rd    7    4   51    0   13 1530    0 0.046728972
## up    4   17    0    0   28    2 1538 0.032095658
```

```r
# confusion matrix
confusionMatrix(preds, test_set$class)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  hm  lm  md  ow  ph  rd  up
##         hm 388   2   0   0   0   3   2
##         lm   4 391   0   0  17   0   5
##         md   0   2 383   1   0  16   0
##         ow   0   0   1 410   0   0   0
##         ph   0   7   0   0 369   2   7
##         rd   1   0  15   0   1 374   0
##         up   0   1   0   0   1   0 397
##
## Overall Statistics
##
##                Accuracy : 0.9686
##                  95% CI : (0.9614, 0.9747)
##     No Information Rate : 0.1468
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9633
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: hm Class: lm Class: md Class: ow Class: ph
## Sensitivity             0.9873    0.9702    0.9599    0.9976    0.9510
## Specificity             0.9971    0.9892    0.9921    0.9996    0.9934
## Pos Pred Value          0.9823    0.9376    0.9527    0.9976    0.9584
## Neg Pred Value          0.9979    0.9950    0.9933    0.9996    0.9921
## Prevalence              0.1404    0.1439    0.1425    0.1468    0.1386
## Detection Rate          0.1386    0.1396    0.1368    0.1464    0.1318
## Detection Prevalence    0.1411    0.1489    0.1436    0.1468    0.1375
## Balanced Accuracy       0.9922    0.9797    0.9760    0.9986    0.9722
##                      Class: rd Class: up
## Sensitivity             0.9468    0.9659
## Specificity             0.9929    0.9992
## Pos Pred Value          0.9565    0.9950
## Neg Pred Value          0.9913    0.9942
## Prevalence              0.1411    0.1468
## Detection Rate          0.1336    0.1418
## Detection Prevalence    0.1396    0.1425
## Balanced Accuracy       0.9699    0.9825
```

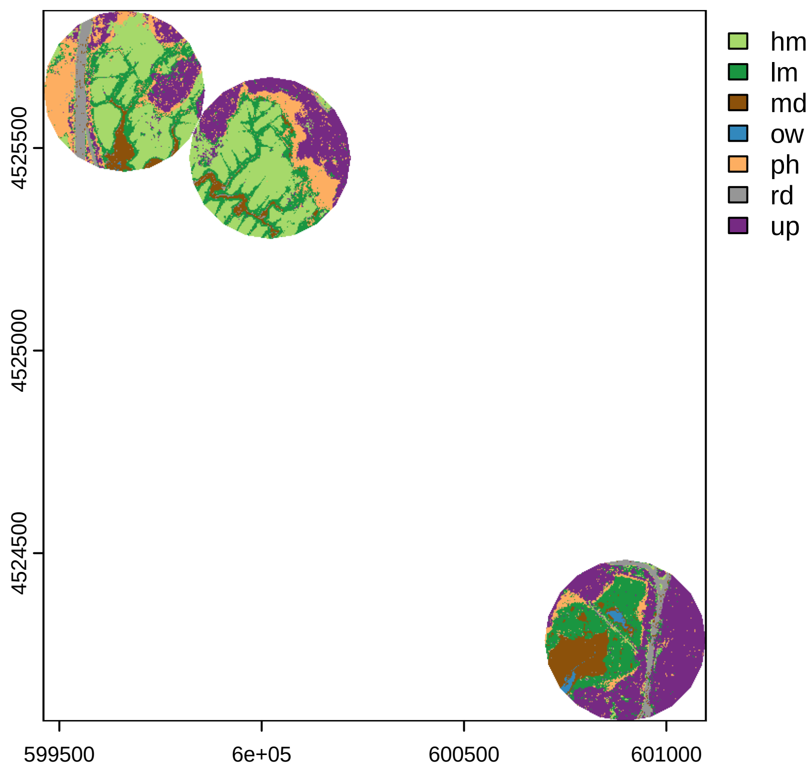
classifying the plots

```r
# load the new, unlabeled raster
prediction_raster <-
↪  rast("~/Desktop/marshbirdsoutput/round_6/prediction_raster_round_6.tif")

# calculate NDVI for prediction raster
ndvi_pred <- (prediction_raster[[4]] - prediction_raster[[1]]) /
  (prediction_raster[[4]] + prediction_raster[[1]])
```

```r
names(ndvi_pred) <- "ndvi"

# calculation NDWI
ndwi_pred <- (prediction_raster[[4]] - prediction_raster[[2]]) /
  (prediction_raster[[4]] + prediction_raster[[2]])
names(ndwi_pred) <- "ndwi"

# Calculate brightness
brightness_pred <- (prediction_raster[[1]] + prediction_raster[[2]] +
                      prediction_raster[[3]] + prediction_raster[[4]]) / 4
names(brightness_pred) <- "brightness"

# Apply the pca
all_for_pca_pred <- c(prediction_raster, ndvi_pred)
names(all_for_pca_pred) <- names(all_for_pca)  # ensure exact match
pca_pred <- predict(all_for_pca_pred, pca_pixels, index = 1:5)
```

```
## |---------|---------|---------|---------|=========================================
```

```r
names(pca_pred) <- paste0("PCA", 1:5)

# stack it
prediction_stack <- c(prediction_raster, ndvi_pred)

# rename to match training names
names(prediction_stack) <- names(r_stack)

# apply the model to predict classes
classified <- predict(prediction_stack, rf_model, na.rm = TRUE)

# Define custom colors
colors <- c(
  "#a6d96a",  # hm
  "#1a9641",  # lm
  "#8c510a",  # md
  "#3288bd",  # ow
  "#fdae61",  # ph
  "#969696",  # rd
  "#762a83"   # up
)



# Plot with colors
plot(classified, col = colors)
```

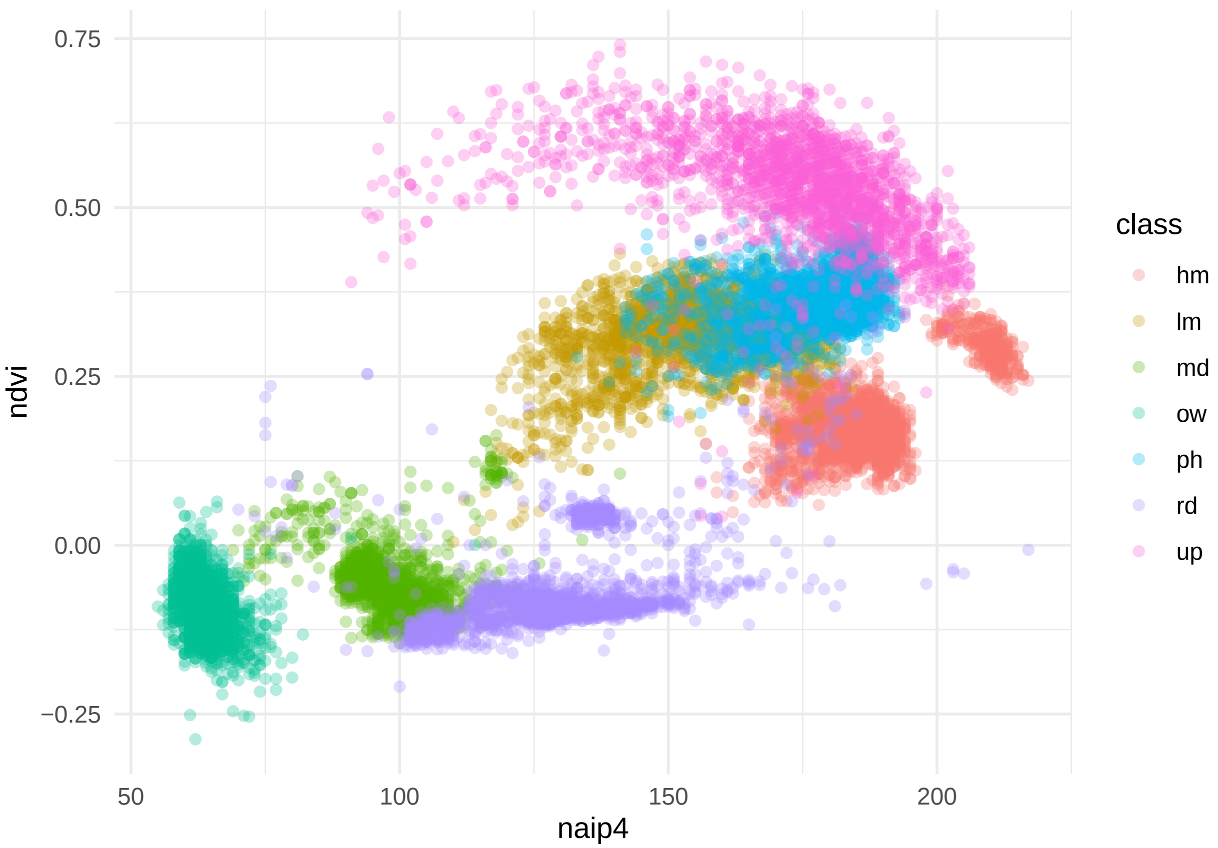



```r
# save the output tif
writeRaster(classified, "~/Desktop/marshbirdsoutput/round_6/classified_output.tif",
↪  overwrite = TRUE)

rf_model$importance
```

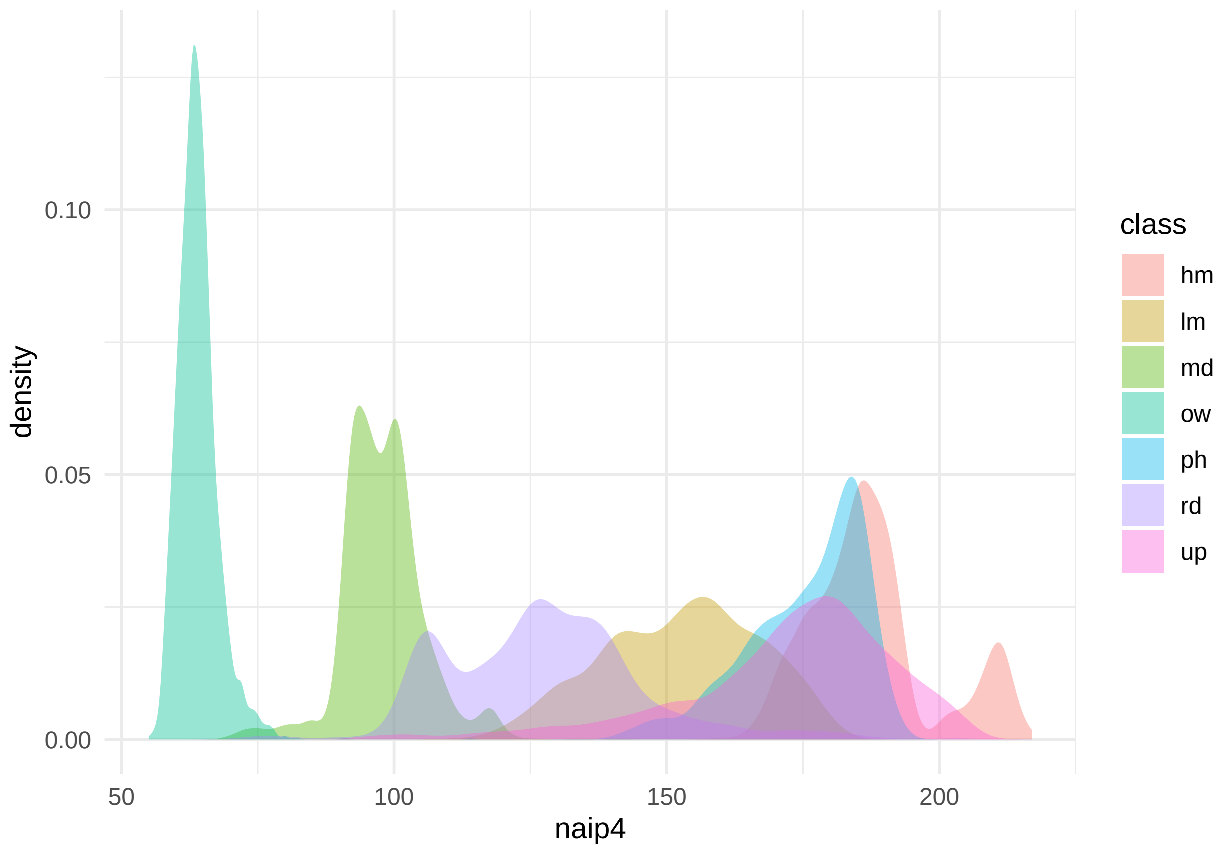
```
##        MeanDecreaseGini
## naip1          1469.464
## naip2          1358.500
## naip3          1927.166
## naip4          2286.801
## ndvi           2549.012
```

feature class importance and visualizations

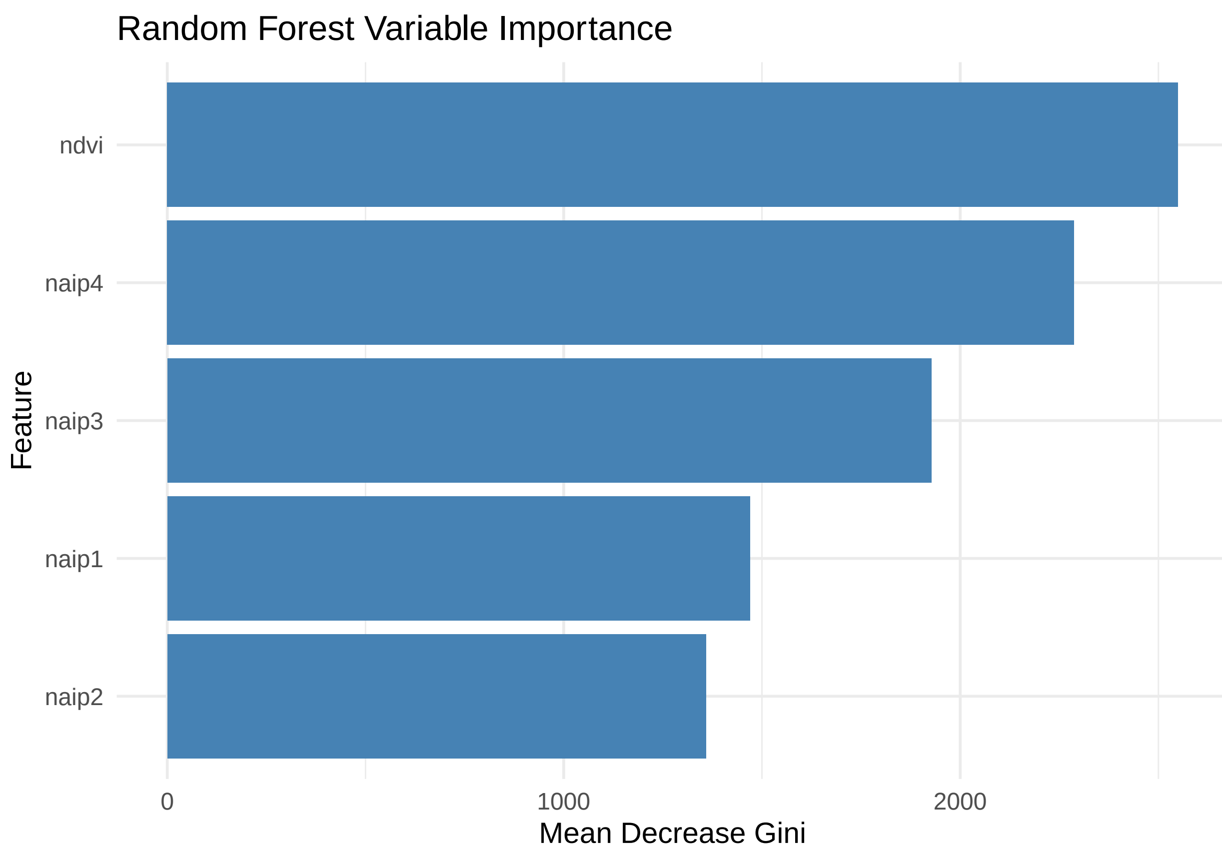
```r
ggplot(training_data, aes(x = naip4, y = ndvi, color = class)) +
  geom_point(alpha = 0.3) +
  theme_minimal()
```




```r
ggplot(training_data, aes(x = naip4, fill = class)) +
  geom_density(alpha = 0.4, color = NA) +
  theme_minimal()
```




```
imp <- as.data.frame(rf_model$importance)
imp$feature <- rownames(imp)

ggplot(imp, aes(x = reorder(feature, MeanDecreaseGini), y = MeanDecreaseGini)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(title = "Random Forest Variable Importance",
       x = "Feature", y = "Mean Decrease Gini") +
  theme_minimal()
```



4. Fourth Iteration: PCA

```
# stack em up
r_stack <- c(pca)

# extract raster values for training polygons
extracted <- terra::extract(r_stack, training_polygons, df = TRUE)

# turn class into a factor
training_polygons$class <- as.factor(training_polygons$class)

# Convert training_polygons to dataframe to get the class labels
poly_df <- as.data.frame(training_polygons)
poly_df$ID <- 1:nrow(poly_df)  # Add ID column to match extract output

# Join the class labels
extracted <- extracted %>%
  left_join(poly_df[, c("ID", "class")], by = "ID")

# clean data by removing rows with na values and remove ID column
extracted_clean <- na.omit(extracted[, -1])  # remove ID and NAs

# sample 2000 per class
```

```r
training_data <- extracted_clean %>%
  group_by(class) %>%
  sample_n(min(2000, n())) %>%  # Use min() to handle small classes
  ungroup()

# turn class column into factor
training_data$class <- factor(training_data$class)
```

check the sampling balance in the classes

print("Class distribution in training data:")

print(table(training_data$class))

```r
# split training and test data
set.seed(342)
idx <- sample(seq_len(nrow(training_data)), size = 0.8 * nrow(training_data))
train_set <- training_data[idx, ]
test_set  <- training_data[-idx, ]

# train random forest model
rf_model <- randomForest(class ~ .,
                         data = train_set,
                         ntree = 500)

# validation metrics
preds <- predict(rf_model, newdata = test_set)
accuracy <- mean(preds == test_set$class)
cat("Accuracy:", round(accuracy, 4), "\n")
```

```
## Accuracy: 0.9779
```

```r
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = class ~ ., data = train_set, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 2.67%
## Confusion matrix:
##      hm   lm   md   ow   ph   rd   up class.error
## hm 1592   12    0    0    0    2    1 0.009334163
## lm   18 1539    5    0   32    0    3 0.036318096
## md    0    2 1554    2    0   43    0 0.029356652
```

```
## ow    0    0    5 1583    0    1    0 0.003775960
## ph    0   44    0    0 1542    1   25 0.043424318
## rd    2    0   40    0   11 1552    0 0.033021807
## up    2   24    0    0   23    1 1539 0.031466331
```

```r
# confusion matrix
confusionMatrix(preds, test_set$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  hm  lm  md  ow  ph  rd  up
##         hm 393   4   0   0   0   0   0
##         lm   0 390   0   0  13   0   2
##         md   0   0 389   4   0   6   0
##         ow   0   0   1 407   0   0   0
##         ph   0   8   1   0 373   4   8
##         rd   0   0   8   0   0 385   0
##         up   0   1   0   0   2   0 401
##
## Overall Statistics
##
##                Accuracy : 0.9779
##                  95% CI : (0.9717, 0.983)
##     No Information Rate : 0.1468
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9742
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: hm Class: lm Class: md Class: ow Class: ph
## Sensitivity             1.0000    0.9677    0.9749    0.9903    0.9613
## Specificity             0.9983    0.9937    0.9958    0.9996    0.9913
## Pos Pred Value          0.9899    0.9630    0.9749    0.9975    0.9467
## Neg Pred Value          1.0000    0.9946    0.9958    0.9983    0.9938
## Prevalence              0.1404    0.1439    0.1425    0.1468    0.1386
## Detection Rate          0.1404    0.1393    0.1389    0.1454    0.1332
## Detection Prevalence    0.1418    0.1446    0.1425    0.1457    0.1407
## Balanced Accuracy       0.9992    0.9807    0.9854    0.9949    0.9763
##                      Class: rd Class: up
## Sensitivity             0.9747    0.9757
## Specificity             0.9967    0.9987
## Pos Pred Value          0.9796    0.9926
## Neg Pred Value          0.9958    0.9958
## Prevalence              0.1411    0.1468
## Detection Rate          0.1375    0.1432
## Detection Prevalence    0.1404    0.1443
## Balanced Accuracy       0.9857    0.9872
```

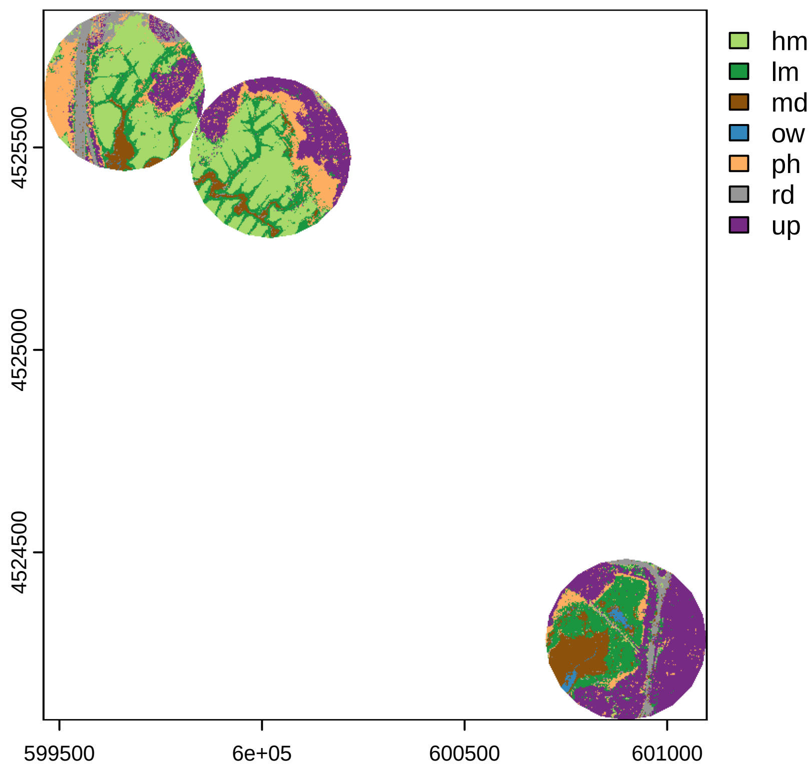
classifying the plots

```r
# load the new, unlabeled raster
prediction_raster <-
↪  rast("~/Desktop/marshbirdsoutput/round_6/prediction_raster_round_6.tif")

# calculate NDVI for prediction raster
ndvi_pred <- (prediction_raster[[4]] - prediction_raster[[1]]) /
  (prediction_raster[[4]] + prediction_raster[[1]])
names(ndvi_pred) <- "ndvi"

# calculation NDWI
ndwi_pred <- (prediction_raster[[4]] - prediction_raster[[2]]) /
  (prediction_raster[[4]] + prediction_raster[[2]])
names(ndwi_pred) <- "ndwi"

# Calculate brightness
brightness_pred <- (prediction_raster[[1]] + prediction_raster[[2]] +
                      prediction_raster[[3]] + prediction_raster[[4]]) / 4
names(brightness_pred) <- "brightness"

# Apply the pca
all_for_pca_pred <- c(prediction_raster, ndvi_pred)
names(all_for_pca_pred) <- names(all_for_pca)  # ensure exact match
pca_pred <- predict(all_for_pca_pred, pca_pixels, index = 1:5)
```

```
## |---------|---------|---------|---------|=========================================
```

```r
names(pca_pred) <- paste0("PCA", 1:5)

# stack it
prediction_stack <- c(pca_pred)

# rename to match training names
names(prediction_stack) <- names(r_stack)

# apply the model to predict classes
classified <- predict(prediction_stack, rf_model, na.rm = TRUE)

# Define custom colors
colors <- c(
  "#a6d96a",  # hm
  "#1a9641",  # lm
  "#8c510a",  # md
  "#3288bd",  # ow
  "#fdae61",  # ph
  "#969696",  # rd
  "#762a83"   # up
)
```

```r
# Plot with colors
plot(classified, col = colors)
```

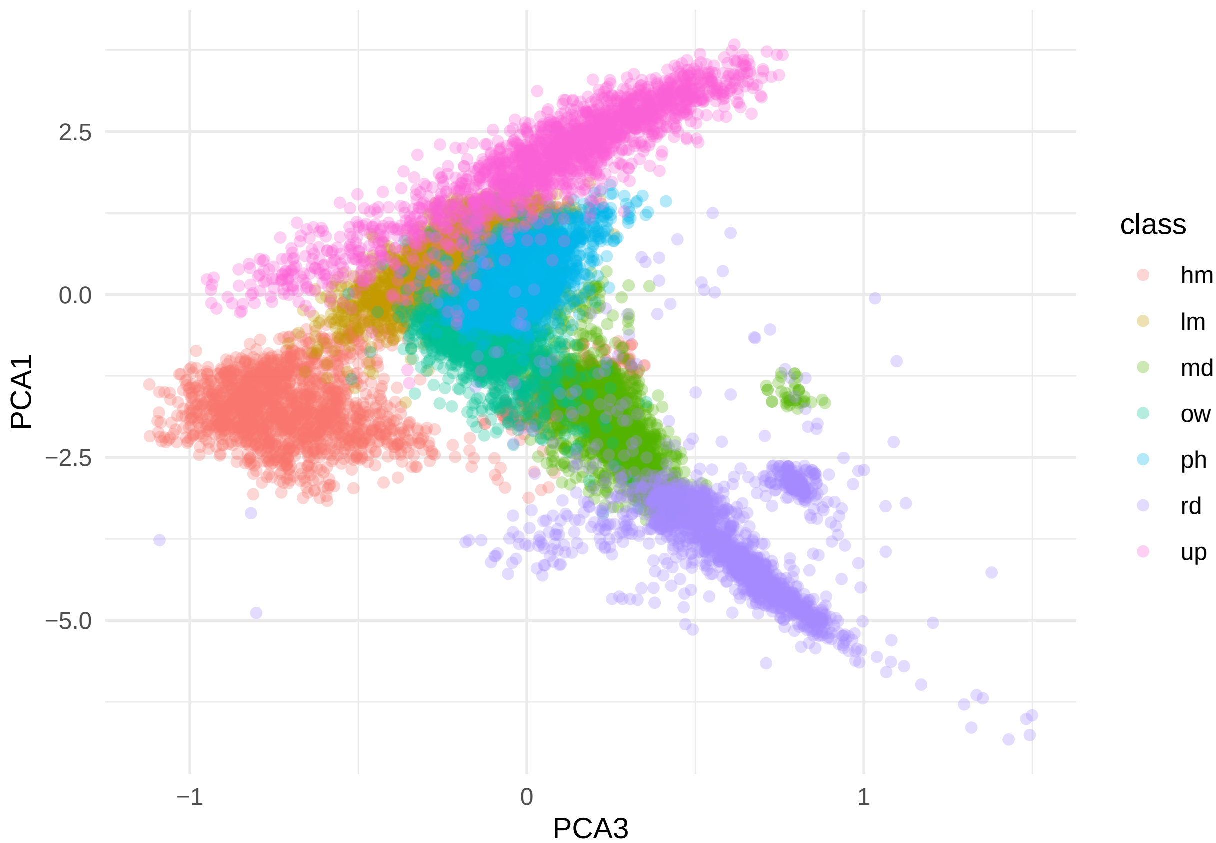



```r
# save the output tif
writeRaster(classified, "~/Desktop/marshbirdsoutput/round_6/classified_output.tif",
↪  overwrite = TRUE)

rf_model$importance
```

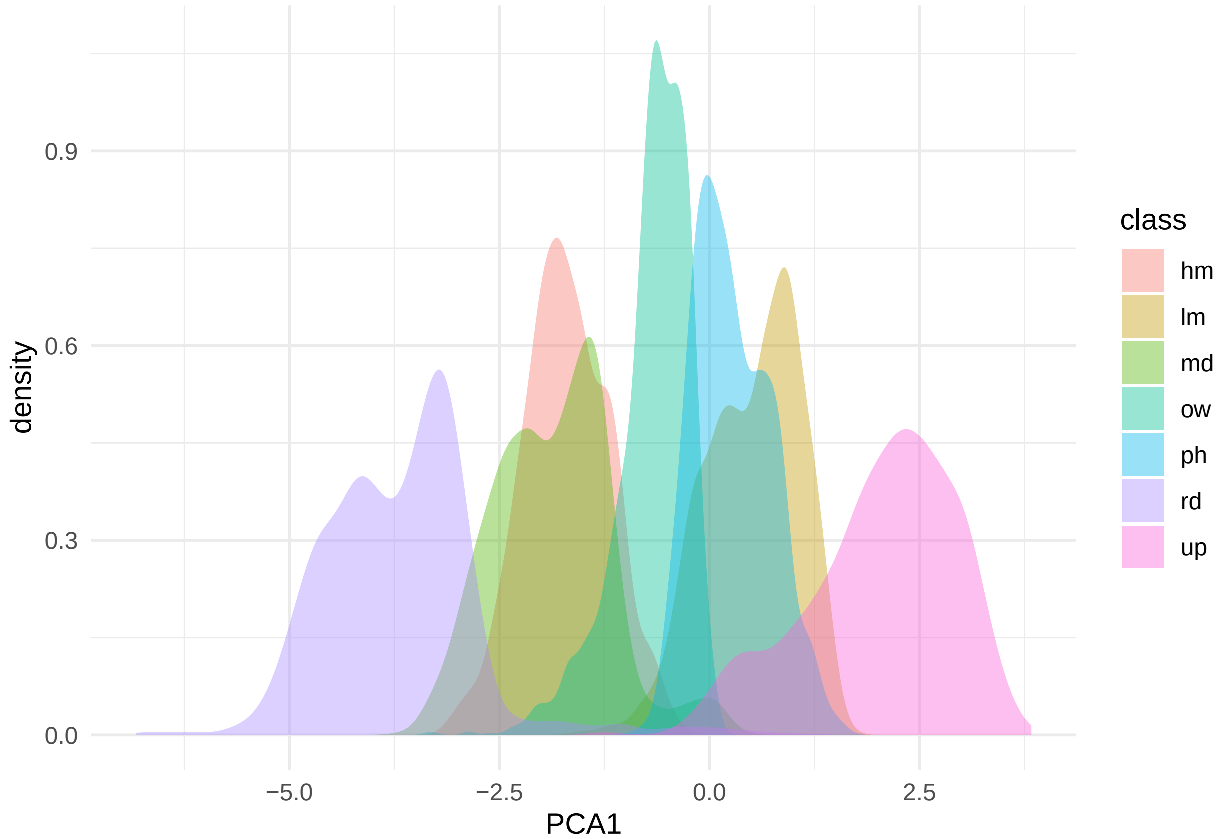
```
##      MeanDecreaseGini
## PCA1        3245.3535
## PCA2        2769.8979
## PCA3        1849.6364
## PCA4         808.6667
## PCA5         925.0009
```

feature class importance and visualizations

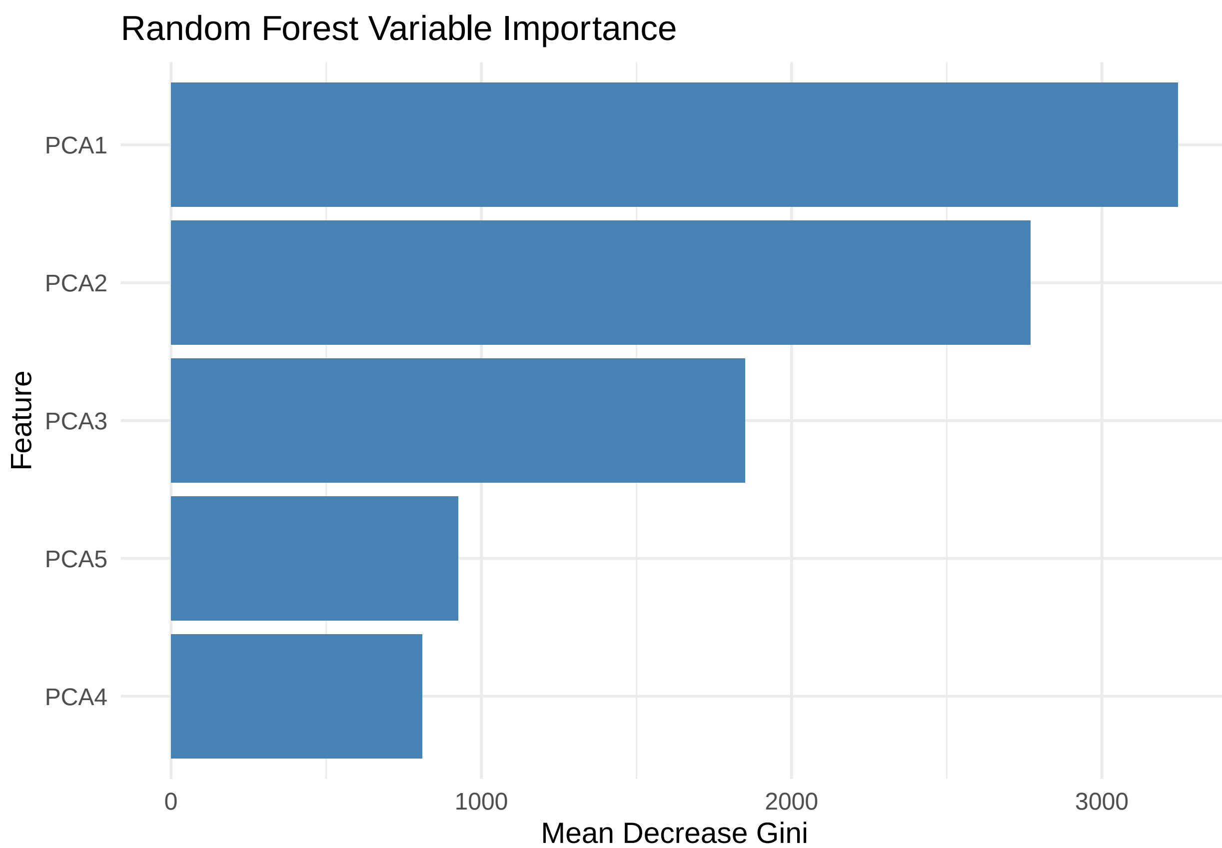
```r
ggplot(training_data, aes(x = PCA3, y = PCA1, color = class)) +
  geom_point(alpha = 0.3) +
  theme_minimal()
```




```r
ggplot(training_data, aes(x = PCA1, fill = class)) +
  geom_density(alpha = 0.4, color = NA) +
  theme_minimal()
```




```r
imp <- as.data.frame(rf_model$importance)
imp$feature <- rownames(imp)

ggplot(imp, aes(x = reorder(feature, MeanDecreaseGini), y = MeanDecreaseGini)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(title = "Random Forest Variable Importance",
       x = "Feature", y = "Mean Decrease Gini") +
  theme_minimal()
```


Random Forest Variable Importance
Feature
PCA1
PCA2
PCA3
PCA5
PCA4
0
1000
2000
3000
Mean Decrease Gini