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_5.tif") # this has the raster data
training_polygons <- vect("training_polygons_round_5.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: 0.9832
```

```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: 1.88%
## Confusion matrix:
##      hm   lm   md   ow   ph   rd   up class.error
## hm 1595   10    0    0    0    2    0 0.007467330
## lm   13 1550    1    0   30    1    2 0.029430182
## md    0    0 1594    0    0    7    0 0.004372267
## ow    0    0    0 1589    0    0    0 0.000000000
## ph    0   42    0    0 1552    3   15 0.037220844
## rd    9    4   19    0   11 1562    0 0.026791277
## up    1   18    0    0   22    1 1547 0.026431718
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  hm  lm  md  ow  ph  rd  up
##         hm 387   4   0   0   0   1   4
##         lm   5 394   0   0  10   0   6
##         md   0   0 398   0   0   3   0
##         ow   0   0   0 411   0   0   0
##         ph   0   4   0   0 377   3   3
##         rd   1   0   1   0   0 388   0
##         up   0   1   0   0   1   0 398
##
## Overall Statistics
##
##                Accuracy : 0.9832
##                  95% CI : (0.9777, 0.9876)
##     No Information Rate : 0.1468
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9804
```

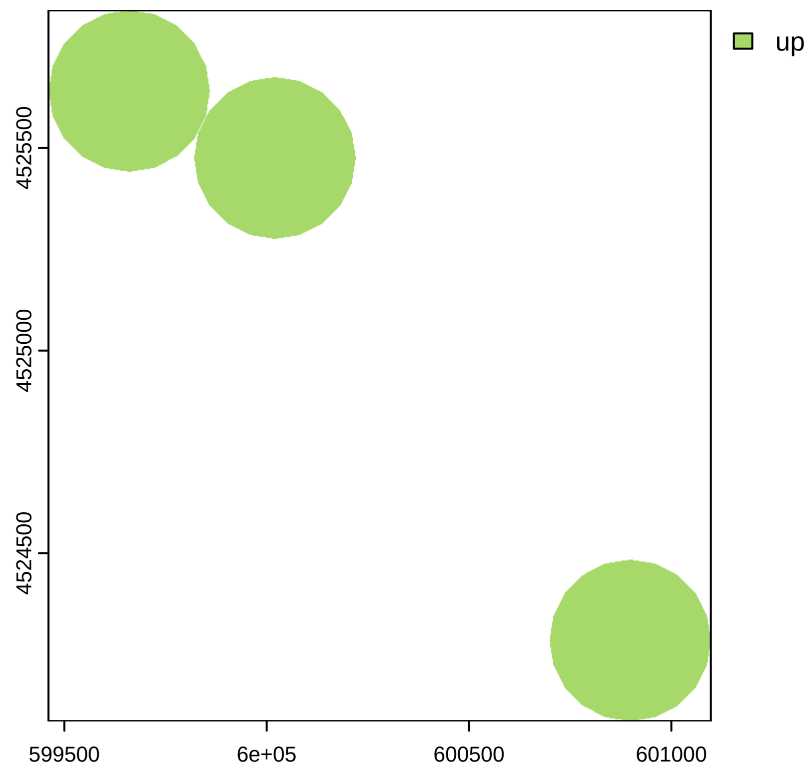
```
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: hm Class: lm Class: md Class: ow Class: ph
## Sensitivity             0.9847    0.9777    0.9975    1.0000    0.9716
## Specificity             0.9963    0.9912    0.9988    1.0000    0.9959
## Pos Pred Value          0.9773    0.9494    0.9925    1.0000    0.9742
## Neg Pred Value          0.9975    0.9962    0.9996    1.0000    0.9954
## Prevalence              0.1404    0.1439    0.1425    0.1468    0.1386
## Detection Rate          0.1382    0.1407    0.1421    0.1468    0.1346
## Detection Prevalence    0.1414    0.1482    0.1432    0.1468    0.1382
## Balanced Accuracy       0.9905    0.9845    0.9981    1.0000    0.9838
##                      Class: rd Class: up
## Sensitivity             0.9823    0.9684
## Specificity             0.9992    0.9992
## Pos Pred Value          0.9949    0.9950
## Neg Pred Value          0.9971    0.9946
## Prevalence              0.1411    0.1468
## Detection Rate          0.1386    0.1421
## Detection Prevalence    0.1393    0.1429
## Balanced Accuracy       0.9907    0.9838
```

classifying the plots

```r
# load the new, unlabeled raster
prediction_raster <-
↪  rast("~/Desktop/marshbirdsoutput/round_5/prediction_raster_round_5.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)
```

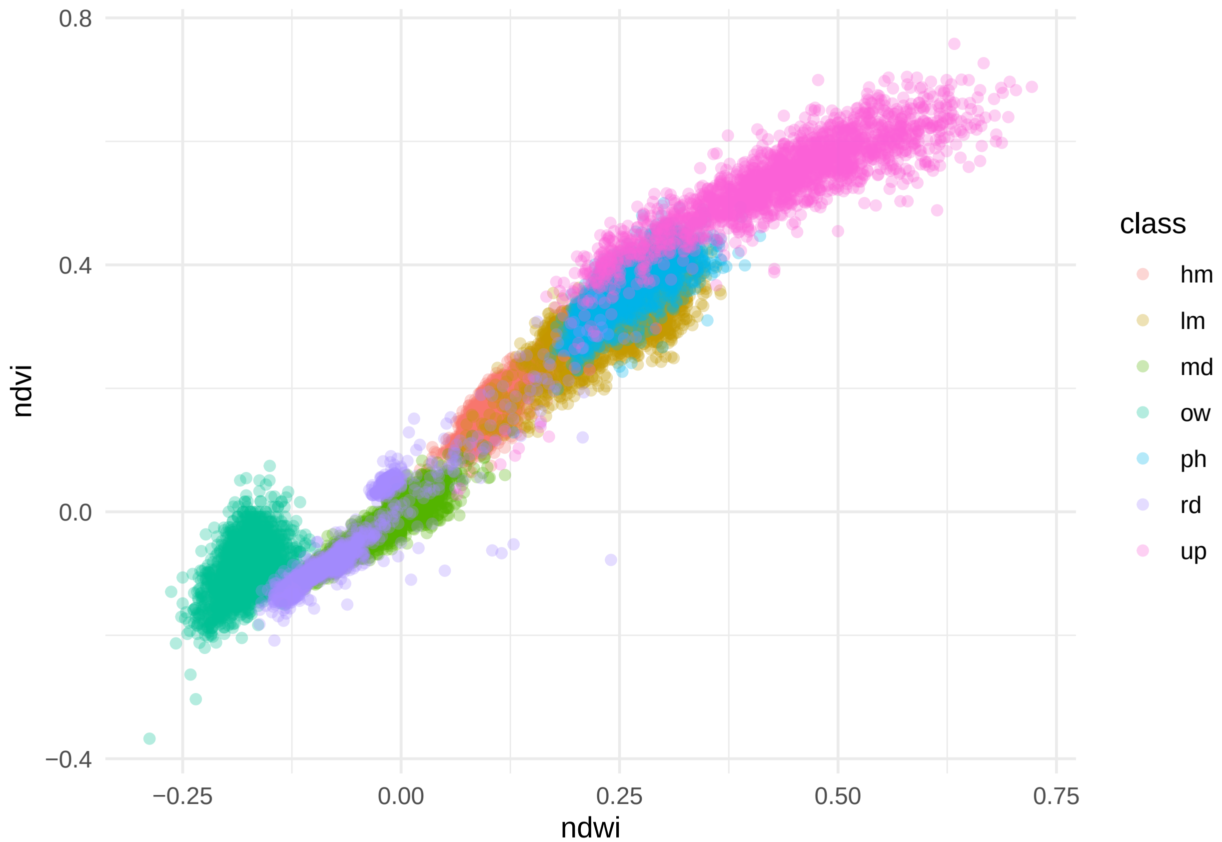



```r
# save the output tif
# writeRaster(classified, "~/Desktop/marshbirdsoutput/round_5/classified_output.tif",
↪  overwrite = TRUE)

rf_model$importance
```

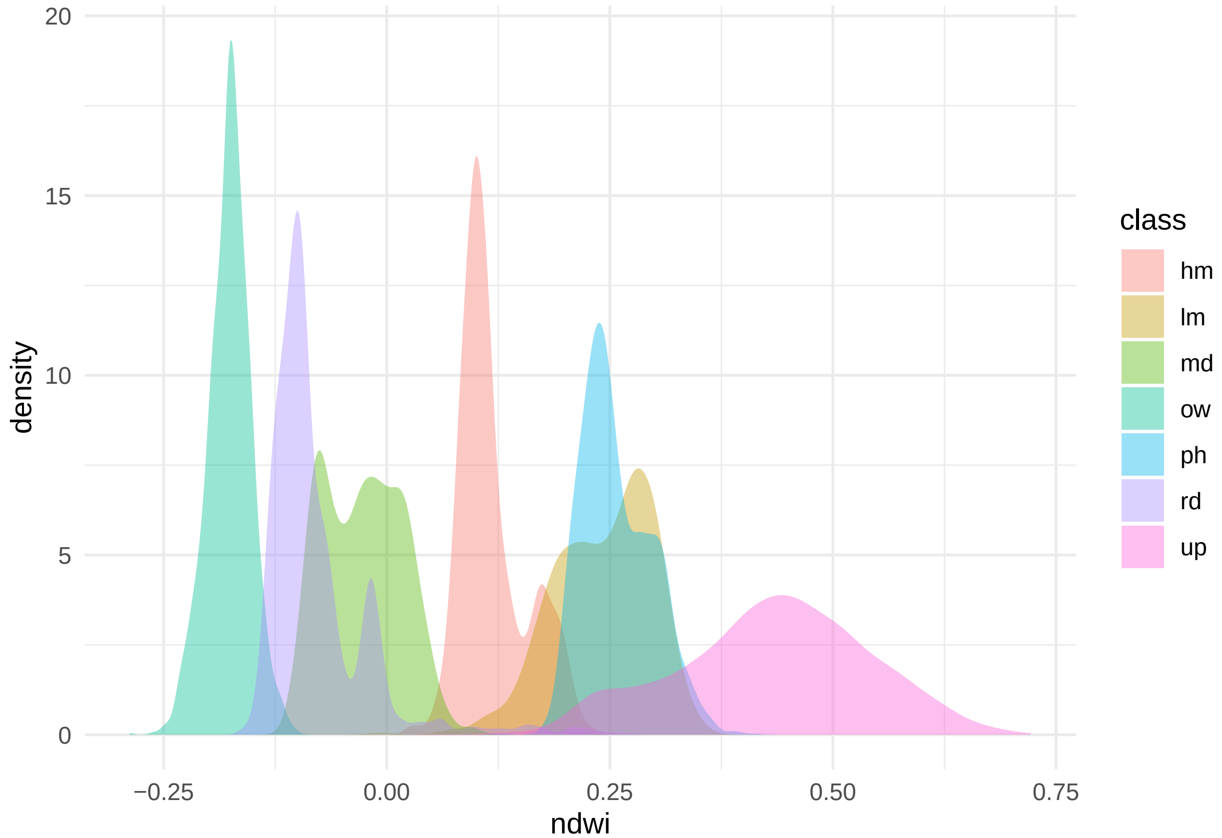
```
##            MeanDecreaseGini
## ndwi              1419.6837
## ndvi              1403.4559
## PCA1               774.4901
## PCA2               613.3142
## PCA3               585.1908
## PCA4               354.1295
## PCA5               404.8675
## naip1              571.6034
## naip2              539.3163
## naip3             1023.2475
## naip4             1089.1408
## brightness         820.6644
```

feature class importance and visualizations

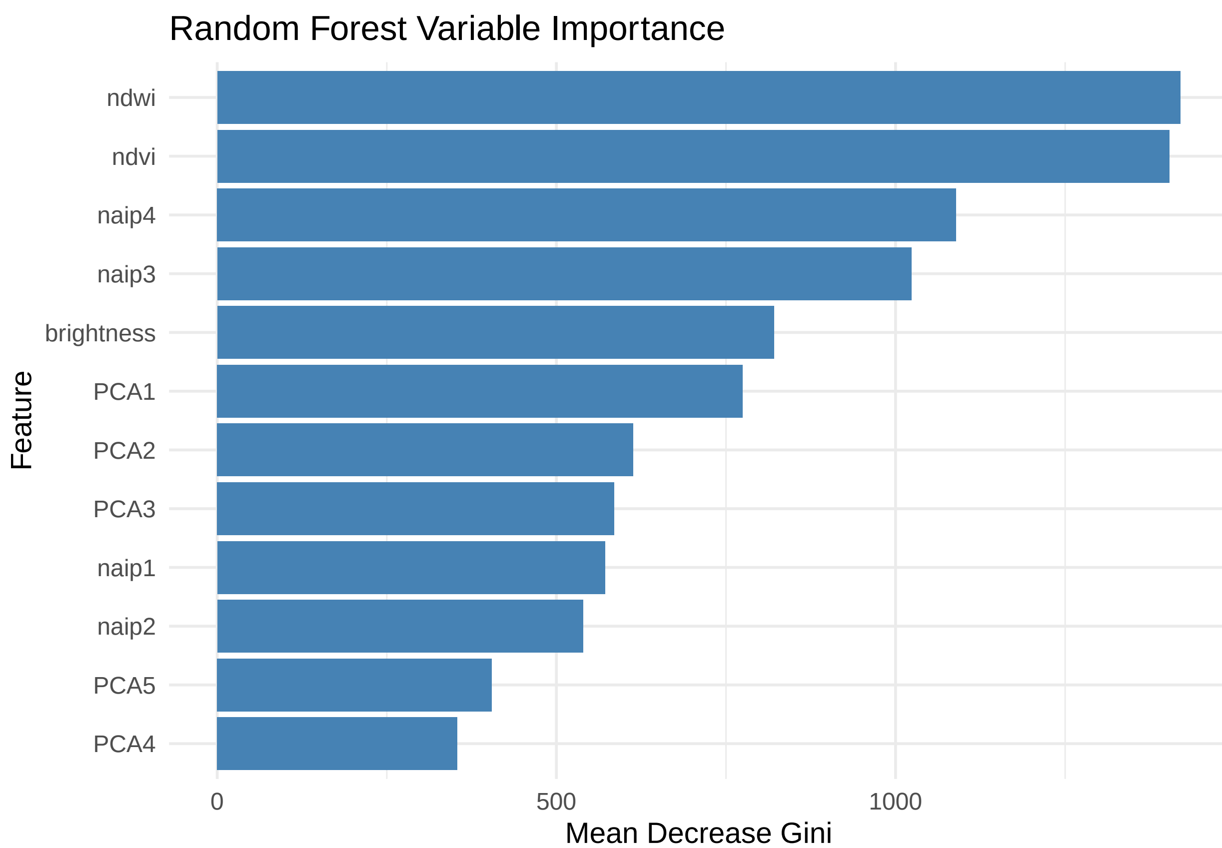
```r
ggplot(training_data, aes(x = ndwi, y = ndvi, color = class)) +
  geom_point(alpha = 0.3) +
  theme_minimal()
```




```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: 0.9814
```

```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: 1.97%
## Confusion matrix:
##      hm   lm   md   ow   ph   rd   up  class.error
## hm 1596    8    0    0    1    2    0 0.0068450529
## lm   20 1544    1    0   29    0    3 0.0331872260
## md    0    1 1595    1    0    4    0 0.0037476577
## ow    0    0    0 1588    0    1    0 0.0006293266
## ph    0   45    0    0 1551    1   15 0.0378411911
## rd    7    2   13    0   10 1573    0 0.0199376947
## up    7   26    0    0   24    0 1532 0.0358716174
```

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

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

```
##           Reference
## Prediction  hm  lm  md  ow  ph  rd  up
##         hm 389   4   0   0   0   1   1
##         lm   4 390   0   0  10   2   4
##         md   0   0 398   0   0   3   0
##         ow   0   0   0 411   0   0   0
##         ph   0   9   0   0 374   2   5
##         rd   0   0   1   0   0 386   1
##         up   0   0   0   0   4   1 400
##
## Overall Statistics
##
##                Accuracy : 0.9814
##                  95% CI : (0.9757, 0.9861)
##     No Information Rate : 0.1468
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9783
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: hm Class: lm Class: md Class: ow Class: ph
## Sensitivity             0.9898    0.9677    0.9975    1.0000    0.9639
## Specificity             0.9975    0.9917    0.9988    1.0000    0.9934
## Pos Pred Value          0.9848    0.9512    0.9925    1.0000    0.9590
## Neg Pred Value          0.9983    0.9946    0.9996    1.0000    0.9942
## Prevalence              0.1404    0.1439    0.1425    0.1468    0.1386
## Detection Rate          0.1389    0.1393    0.1421    0.1468    0.1336
## Detection Prevalence    0.1411    0.1464    0.1432    0.1468    0.1393
## Balanced Accuracy       0.9937    0.9797    0.9981    1.0000    0.9786
##                      Class: rd Class: up
## Sensitivity             0.9772    0.9732
## Specificity             0.9992    0.9979
## Pos Pred Value          0.9948    0.9877
## Neg Pred Value          0.9963    0.9954
## Prevalence              0.1411    0.1468
## Detection Rate          0.1379    0.1429
## Detection Prevalence    0.1386    0.1446
## Balanced Accuracy       0.9882    0.9856
```

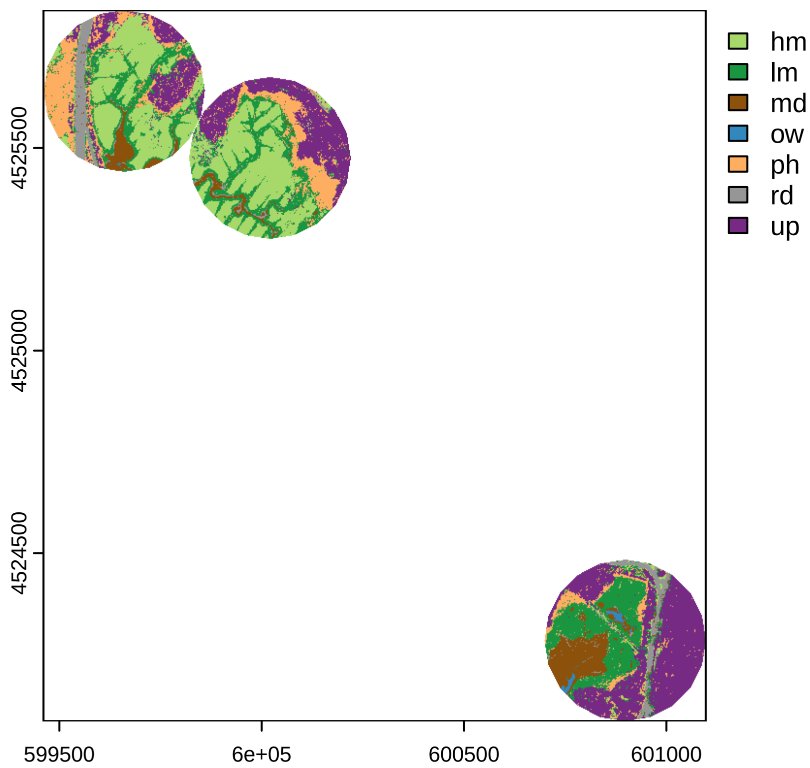
classifying the plots

```
# load the new, unlabeled raster
prediction_raster <-
↪  rast("~/Desktop/marshbirdsoutput/round_5/prediction_raster_round_5.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)
```

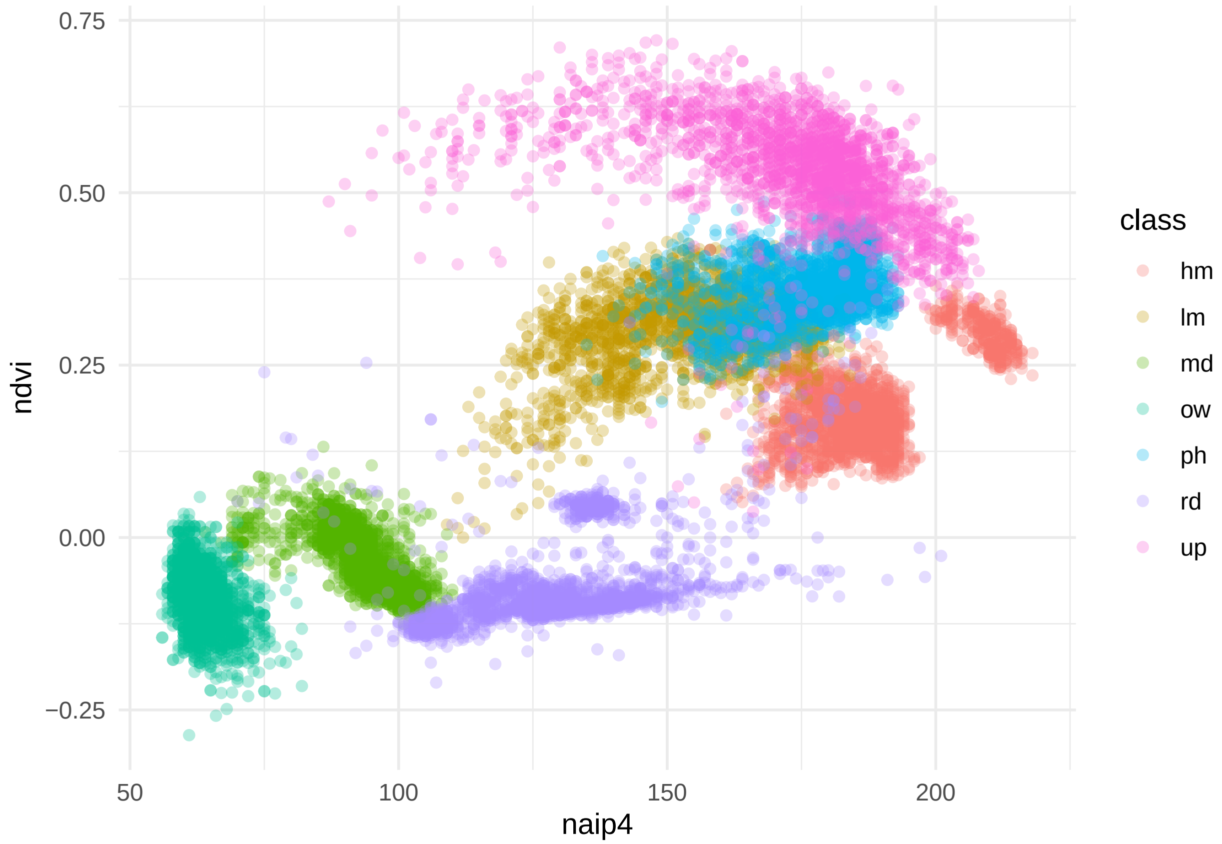



```r
# save the output tif
# writeRaster(classified, "~/Desktop/marshbirdsoutput/round_5/classified_output.tif",
↪  overwrite = TRUE)

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

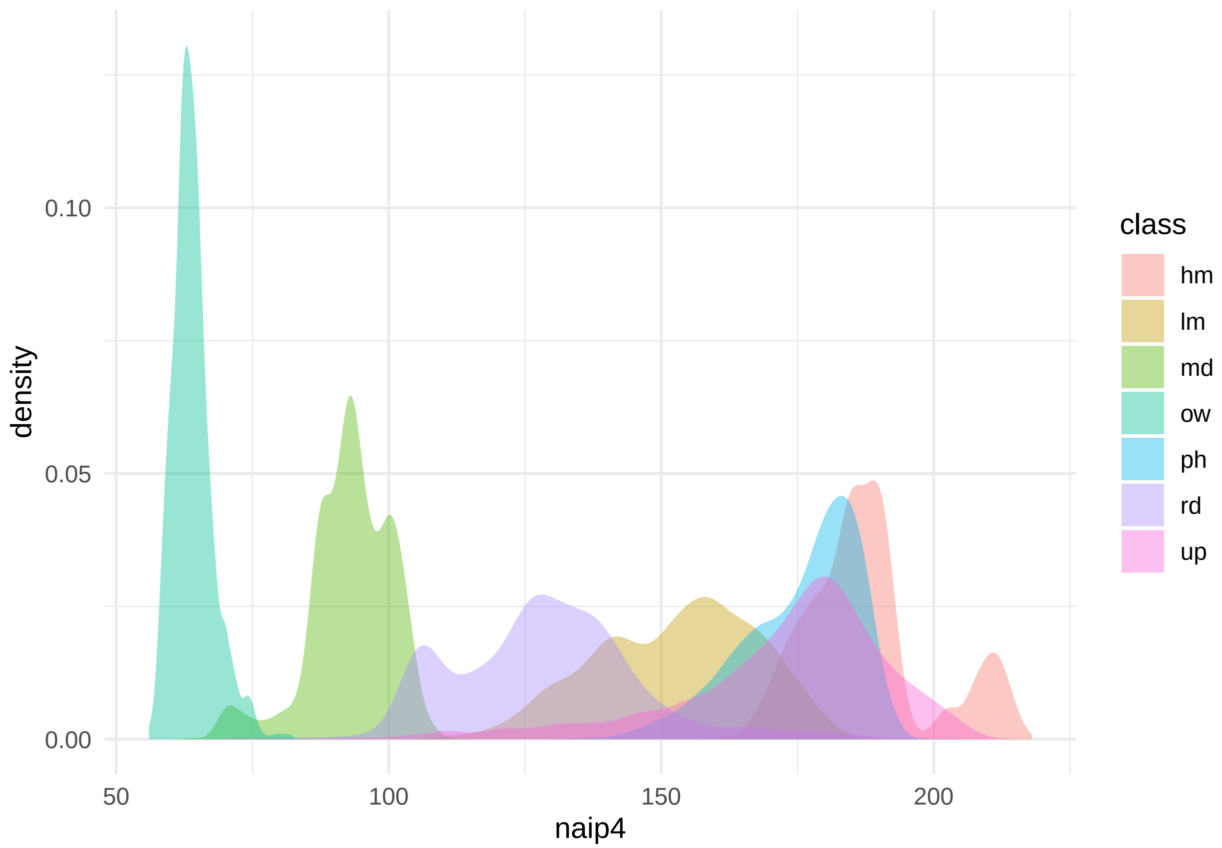
```
##       MeanDecreaseGini
## naip1         1183.124
## naip2         1291.161
## naip3         1724.649
## naip4         1585.792
## ndvi          1803.991
## ndwi          2005.977
```

feature class importance and visualizations

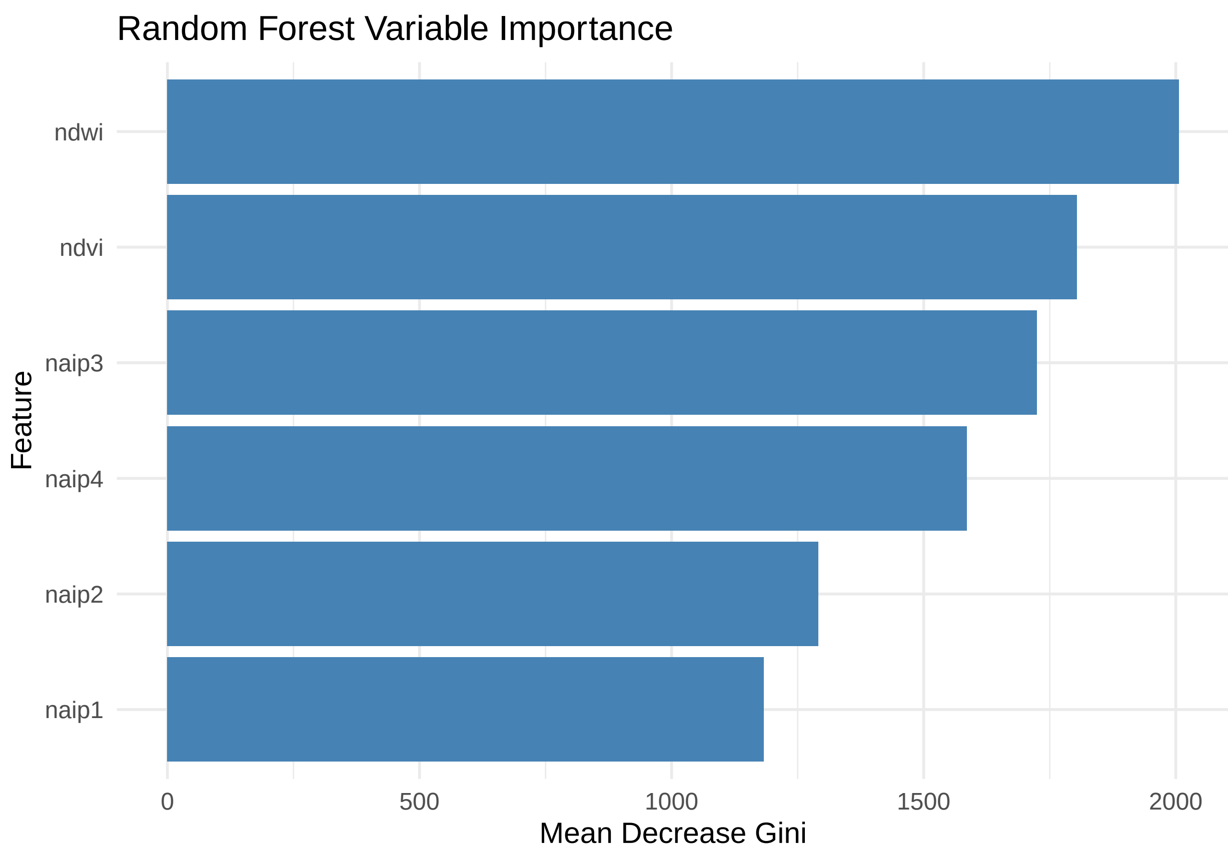
```r
ggplot(training_data, aes(x = naip4, y = ndvi, color = class)) +
  geom_point(alpha = 0.3) +
  theme_minimal()
```




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




```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: 0.9821
```

```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: 2.26%
## Confusion matrix:
##      hm   lm   md   ow   ph   rd   up class.error
## hm 1597   10    0    0    0    0    0 0.006222775
## lm   15 1538    0    0   40    3    1 0.036944271
## md    0    0 1594    0    0    7    0 0.004372267
## ow    0    0    1 1587    0    1    0 0.001258653
## ph    1   61    0    0 1537    3   10 0.046526055
## rd    9    4   23    0   11 1558    0 0.029283489
## up    2   26    0    0   22    3 1536 0.033354311
```

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

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

```
##           Reference
## Prediction  hm  lm  md  ow  ph  rd  up
##         hm 391   5   0   0   0   0   1
##         lm   2 387   0   0  11   1   4
##         md   0   0 399   0   0   3   0
##         ow   0   0   0 411   0   0   0
##         ph   0   9   0   0 376   6   4
##         rd   0   1   0   0   0 385   1
##         up   0   1   0   0   1   0 401
##
## Overall Statistics
##
##                Accuracy : 0.9821
##                  95% CI : (0.9765, 0.9867)
##     No Information Rate : 0.1468
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9792
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: hm Class: lm Class: md Class: ow Class: ph
## Sensitivity             0.9949    0.9603    1.0000    1.0000    0.9691
## Specificity             0.9975    0.9925    0.9988    1.0000    0.9921
## Pos Pred Value          0.9849    0.9556    0.9925    1.0000    0.9519
## Neg Pred Value          0.9992    0.9933    1.0000    1.0000    0.9950
## Prevalence              0.1404    0.1439    0.1425    0.1468    0.1386
## Detection Rate          0.1396    0.1382    0.1425    0.1468    0.1343
## Detection Prevalence    0.1418    0.1446    0.1436    0.1468    0.1411
## Balanced Accuracy       0.9962    0.9764    0.9994    1.0000    0.9806
##                      Class: rd Class: up
## Sensitivity             0.9747    0.9757
## Specificity             0.9992    0.9992
## Pos Pred Value          0.9948    0.9950
## Neg Pred Value          0.9959    0.9958
## Prevalence              0.1411    0.1468
## Detection Rate          0.1375    0.1432
## Detection Prevalence    0.1382    0.1439
## Balanced Accuracy       0.9869    0.9874
```

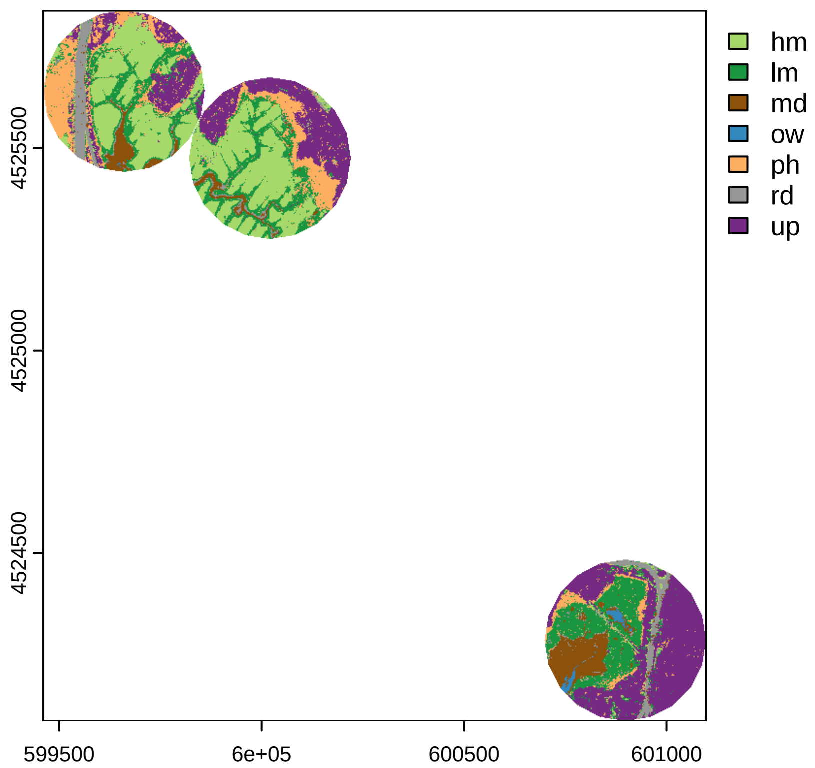
classifying the plots

```r
# load the new, unlabeled raster
prediction_raster <-
↪  rast("~/Desktop/marshbirdsoutput/round_5/prediction_raster_round_5.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)
```

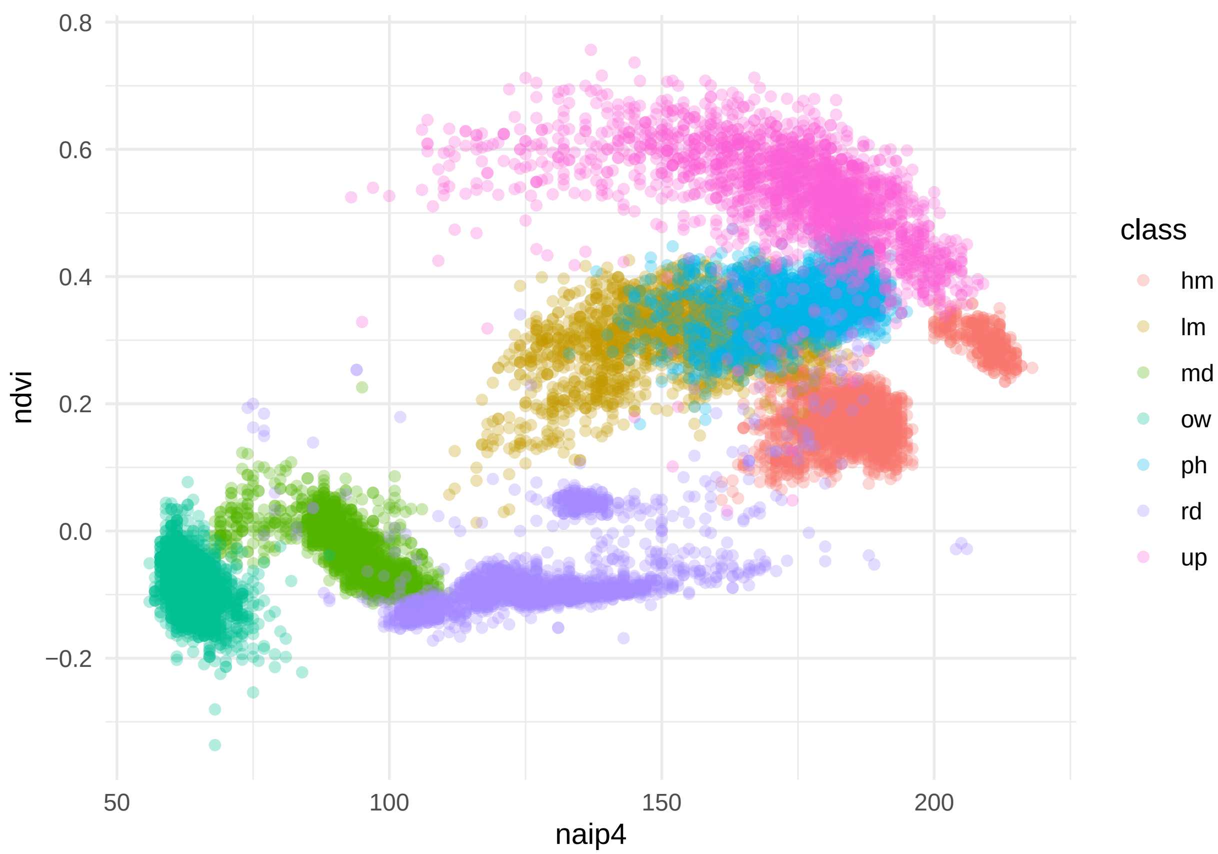



```r
# save the output tif
writeRaster(classified, "~/Desktop/marshbirdsoutput/round_5/classified_output.tif",
↪  overwrite = TRUE)

rf_model$importance
```

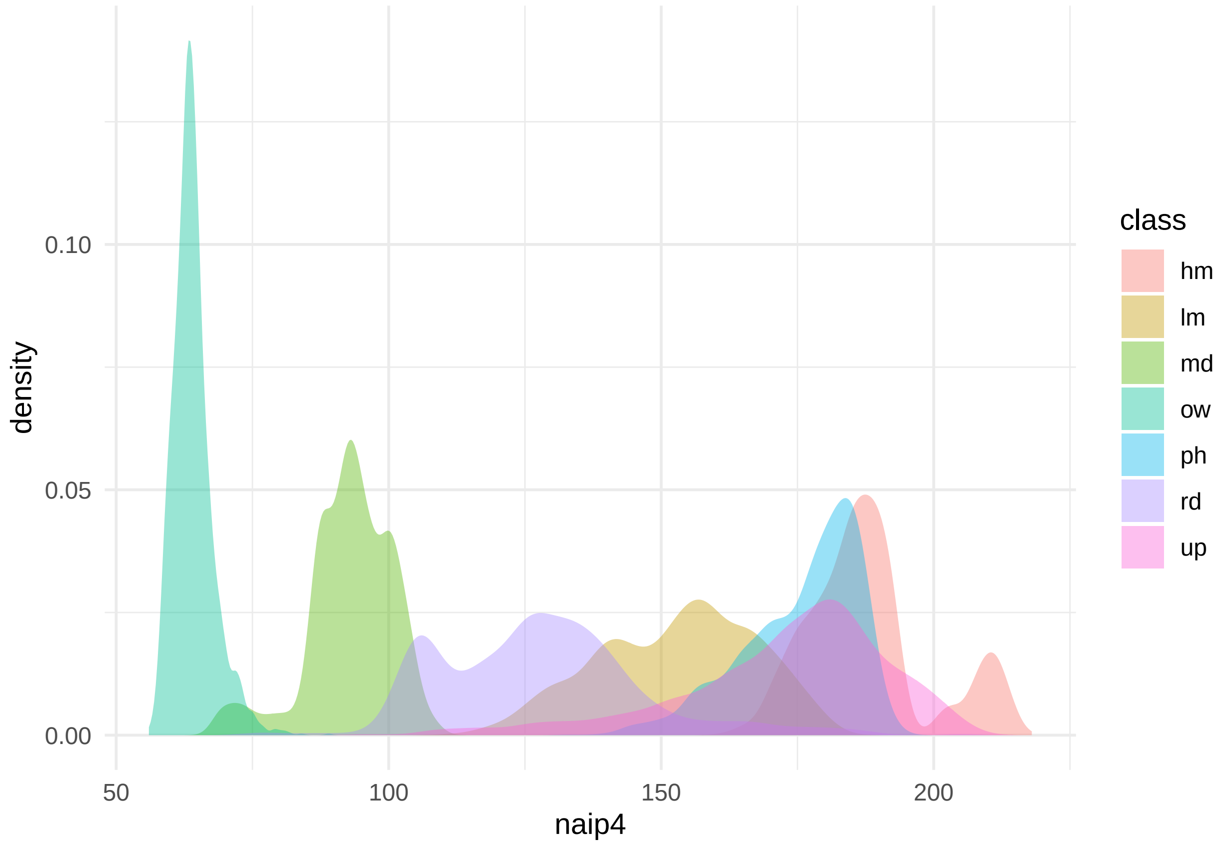
```
##        MeanDecreaseGini
## naip1          1461.871
## naip2          1427.051
## naip3          1937.398
## naip4          2257.483
## ndvi           2510.155
```

feature class importance and visualizations

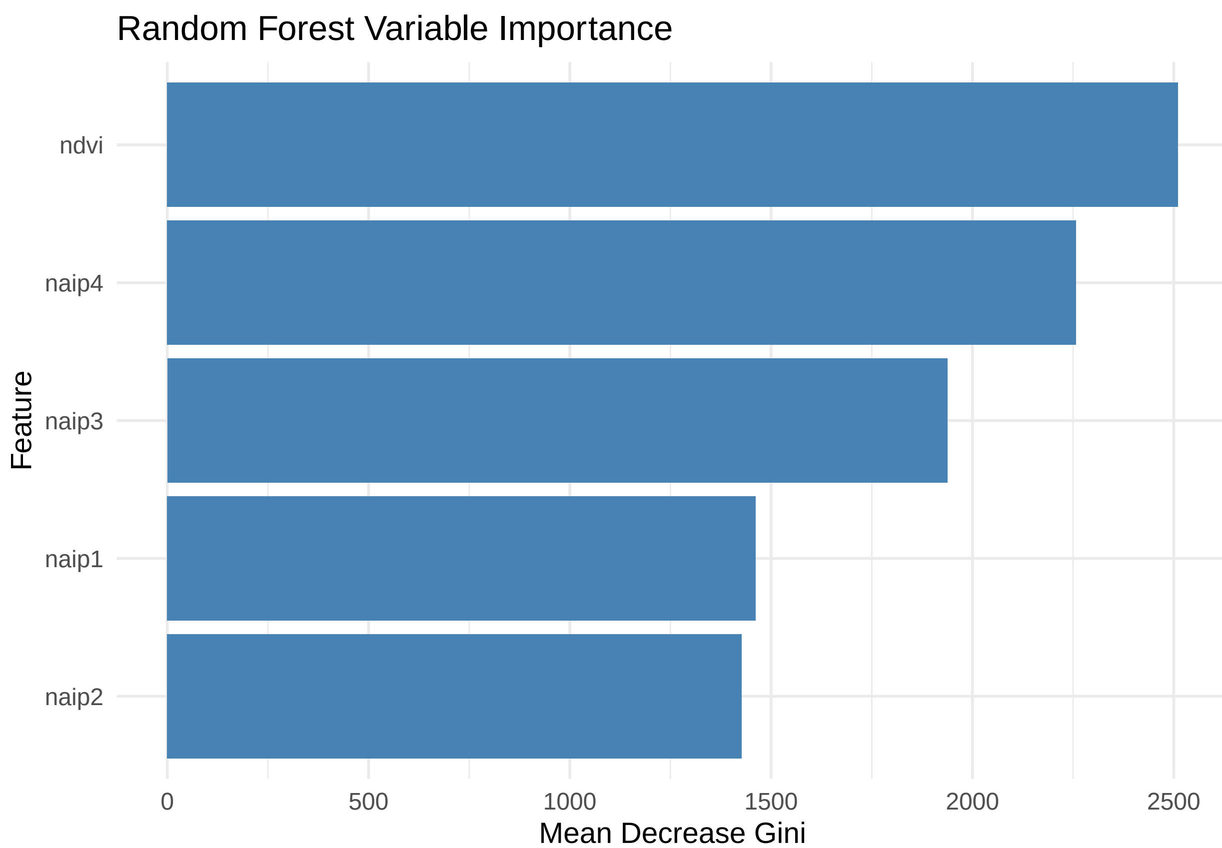
```r
ggplot(training_data, aes(x = naip4, y = ndvi, color = class)) +
  geom_point(alpha = 0.3) +
  theme_minimal()
```




```
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()
```



4. Fourth Iteration: PCA

```r
# 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: 0.9771
```

```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: 1.78%
## Confusion matrix:
##      hm   lm   md   ow   ph   rd   up class.error
## hm 1596    9    0    0    1    1    0 0.006845053
## lm   10 1550    7    0   25    1    4 0.029430182
## md    0    0 1591    2    0    8    0 0.006246096
```

```
## ow    0    0    8 1579    0    2    0 0.006293266
## ph    0   40    0    0 1555    1   16 0.035359801
## rd    0    0   13    1   11 1579    1 0.016199377
## up    0   12    0    0   26    0 1551 0.023914412
```

```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   0   1
##         lm   4 387   1   0  15   0   5
##         md   0   1 394   3   0   1   0
##         ow   0   0   2 408   0   0   0
##         ph   0   9   0   0 368   1   7
##         rd   0   0   2   0   1 393   0
##         up   1   2   0   0   4   0 398
##
## Overall Statistics
##
##                Accuracy : 0.9771
##                  95% CI : (0.9709, 0.9824)
##     No Information Rate : 0.1468
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9733
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: hm Class: lm Class: md Class: ow Class: ph
## Sensitivity             0.9873    0.9603    0.9875    0.9927    0.9485
## Specificity             0.9979    0.9896    0.9979    0.9992    0.9930
## Pos Pred Value          0.9873    0.9393    0.9875    0.9951    0.9558
## Neg Pred Value          0.9979    0.9933    0.9979    0.9987    0.9917
## Prevalence              0.1404    0.1439    0.1425    0.1468    0.1386
## Detection Rate          0.1386    0.1382    0.1407    0.1457    0.1314
## Detection Prevalence    0.1404    0.1471    0.1425    0.1464    0.1375
## Balanced Accuracy       0.9926    0.9749    0.9927    0.9959    0.9707
##                      Class: rd Class: up
## Sensitivity             0.9949    0.9684
## Specificity             0.9988    0.9971
## Pos Pred Value          0.9924    0.9827
## Neg Pred Value          0.9992    0.9946
## Prevalence              0.1411    0.1468
## Detection Rate          0.1404    0.1421
## Detection Prevalence    0.1414    0.1446
## Balanced Accuracy       0.9968    0.9827
```

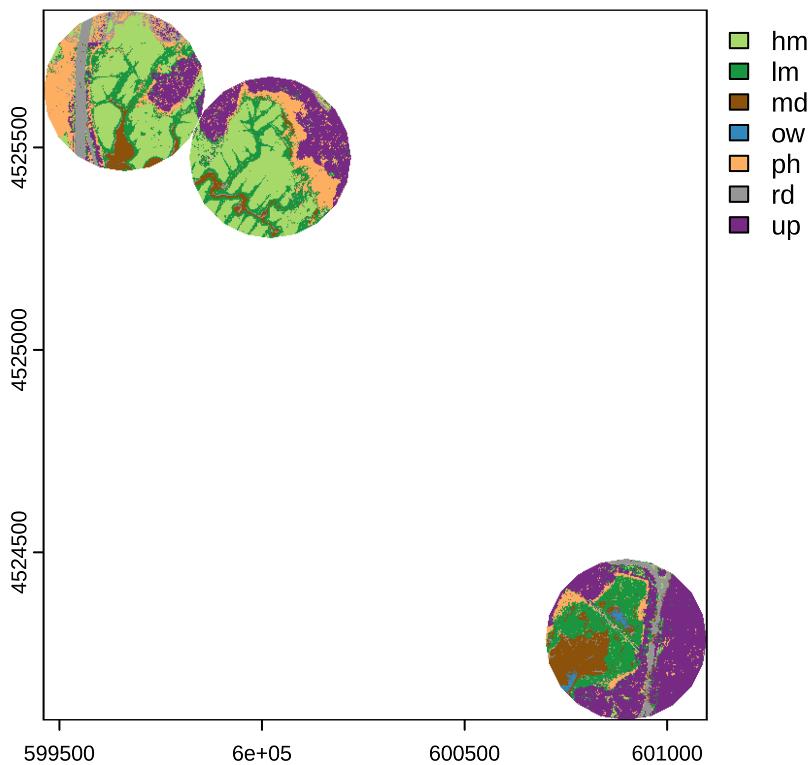
classifying the plots

```
# load the new, unlabeled raster
prediction_raster <-
↪  rast("~/Desktop/marshbirdsoutput/round_5/prediction_raster_round_5.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)
```

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

```
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)
```

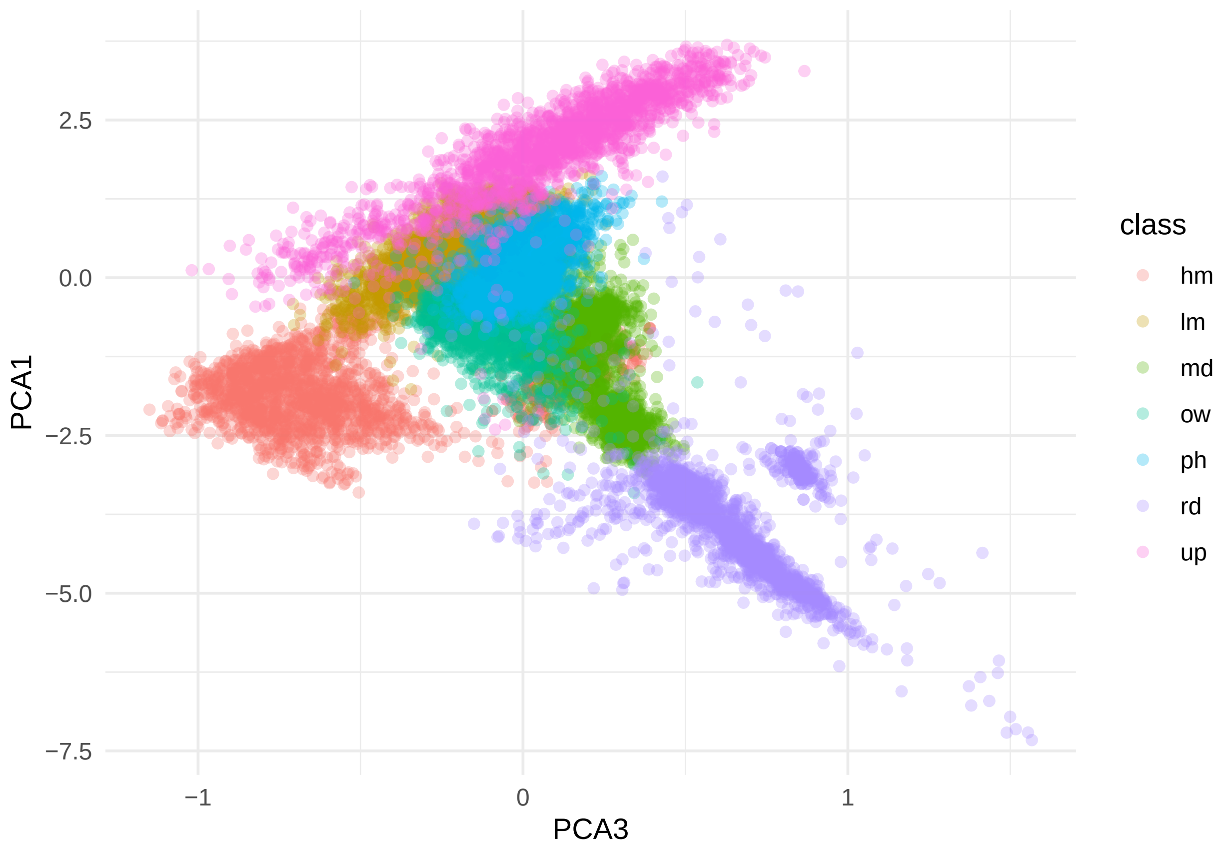



```r
# save the output tif
writeRaster(classified, "~/Desktop/marshbirdsoutput/round_5/classified_output.tif",
↪  overwrite = TRUE)

rf_model$importance
```

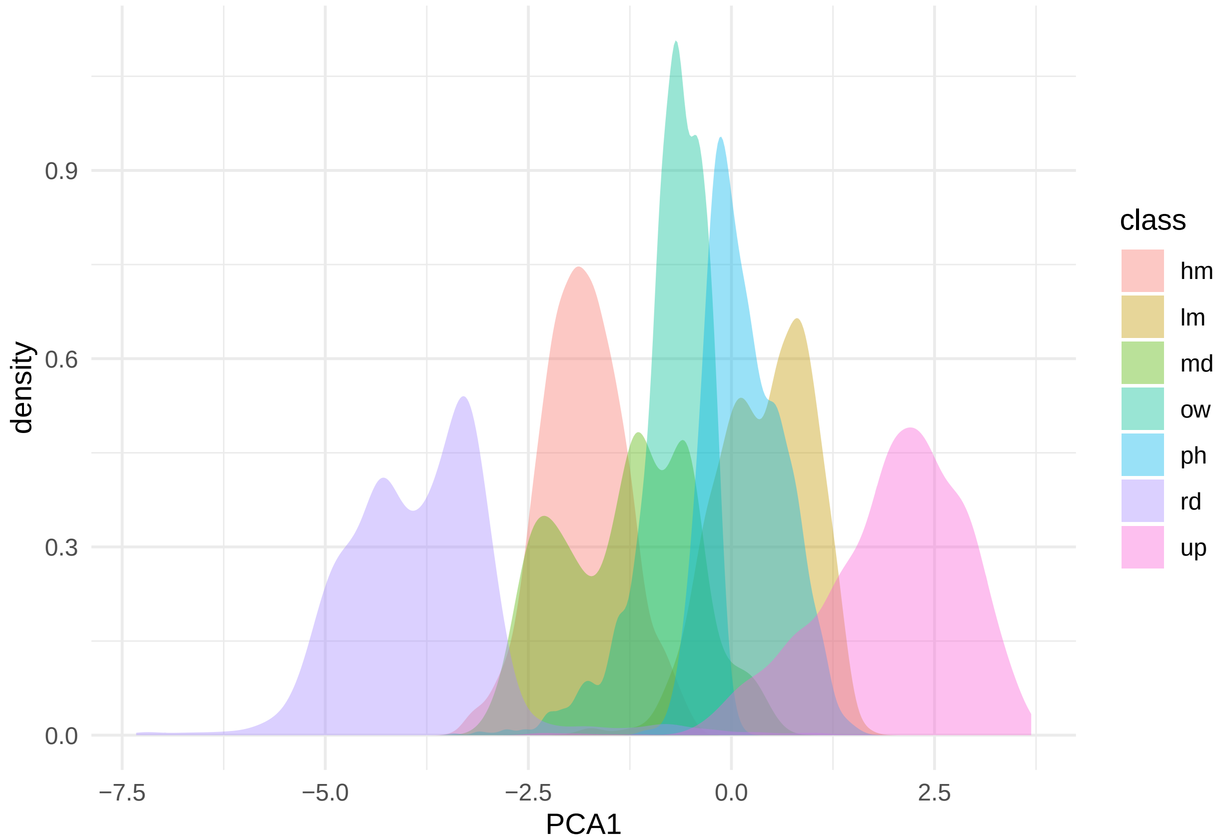
```
##      MeanDecreaseGini
## PCA1        3351.2342
## PCA2        2768.7772
## PCA3        1786.6728
## PCA4         838.4614
## PCA5         854.0149
```

feature class importance and visualizations

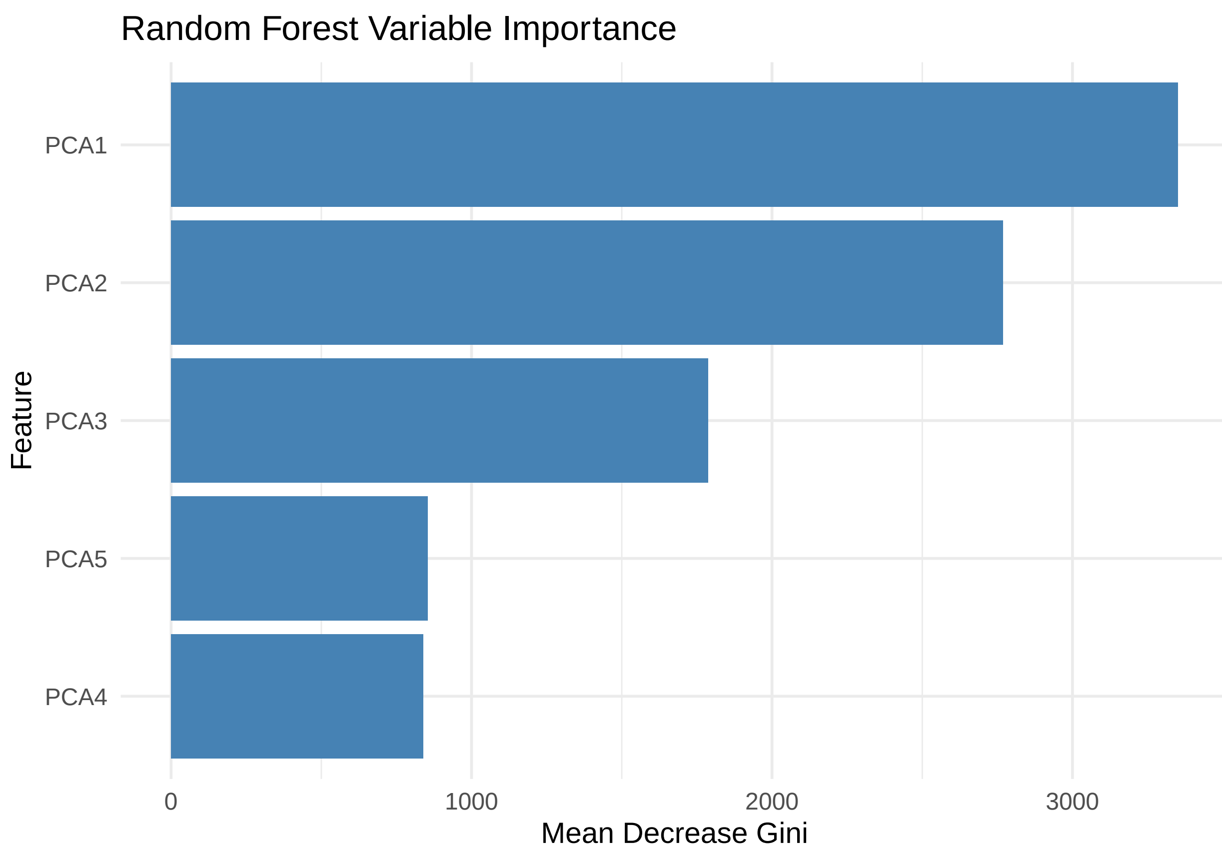
```r
ggplot(training_data, aes(x = PCA3, y = PCA1, color = class)) +
  geom_point(alpha = 0.3) +
  theme_minimal()
```




```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
PCA1
PCA2
PCA3
PCA5
PCA4
Feature
0
1000
2000
3000
Mean Decrease Gini