Predictive Modeling for Crohn's Disease and Ulcerative Colitis

Jacob Halle and Destiny Collazo

Abstract

Crohn's Disease and Ulcerative Colitis are the two most prevalent inflammatory bowel diseases in the United States, yet diagnosis of these conditions remains challenging. In this paper, we develop a variety of machine learning models to classify patients as having Crohn's Disease, Ulcerative Colitis, or healthy control given transcriptomic data and clinical features. Support vector machine (SVM), decision tree, random forest, adaboost, and xgboost models were trained on a data set containing 2,471 total participants[1]. Differentially expressed gene (DEG) analysis was performed on the two diseases to identify genes likely to be predictive. 30-40 of the most significant DEGs across both diseases were used to train the models, as well as selected metadata such as age, sex, and biopsy region. Each model was evaluated on accuracy, precision, recall, and F1 at a set threshold, as well as AUC. We identified Adaboost as the model with the highest overall accuracy with 82% when considering all three classes. SVM, random forest, and adaboost had the highest average AUC, suggesting these models are the best descriminators. Future studies including a larger variety of conditions affecting the gut are necessary to evaluate the feasibility of our machine learning models in diagnostic settings. The initial results of our models highlight the potential of machine learning models to enhance diagnostic accuracy.

Clear Statement of Research Question

Can a machine learning classifier aid in the diagnosis of Crohn's disease and Ulcerative Colitis using gene expression data?

Hypothesis

A machine learning classifier can accurately distinguish patients with Crohn's disease, ulcerative colitis, or healthy status when provided with biomarker and demographic information.

Background

Inflammatory bowel disease (IBD) categorizes a variety of autoimmune conditions that occur when the host's immune system promotes chronic inflammation in the gastrointestinal (GI) tract. The two most common forms of IBD are Ulcerative colitis (UC), which affects the colon and rectum, and Crohn's disease (CD), which can affect the entire GI tract and often results in sparse patches of inflammation. Although both conditions have overlapping symptoms and phenotype, they are distinct conditions with varying patterns, complications, and depth of tissue layer impact. As a result, IBD is notorious for being difficult to diagnose as well as differentiate the two diseases, which is crucial for adequate treatment. A combination of diet, environmental factors, and genetics lead to a variety of symptoms, severity, and even microbiome, making its presentation almost unique to each patient. Several factors contribute to IBD prognosis including age and gender; as there is a higher occurrence of Crohn's diagnoses before 30 years of age, and women are more likely to develop Crohn's but men are more likely to develop ulcerative colitis.

The aim of this project is to ascertain whether machine learning can be a tool for assisting, even expediting, the diagnosis process.

A combination of methods are currently used to diagnose IBD, including imaging studies, stool samples, and blood tests. Biomarkers include C-reactive protein, which can be elevated in the blood during inflammation, and fecal calprotectin, a protein in white blood cells which can be elevated during IBD. However, neither of these methods are precise enough to adequately diagnose or predict IBD presence or prognosis. A patient's symptoms can often overlap with other potential GI diagnoses, as well as between the 2 diseases in question, further complicating the matter of prompt and accurate diagnosis. While both diseases result in inflammation of the GI tract, the genes and mechanisms underlying each disease differ, requiring an accurate diagnosis to begin the correct treatment. Due to the limited precision of the current tests for IBD, approximately 15% of UC cases are eventually determined to be CD[2] and the diagnosis is switched. When diagnosed, several scoring systems are used to assess quality of life and activity level of the disease based on the patient's symptoms, pain level, bowel movements, and/or endoscopic appearance of certain structures. A few include the Harvey-Bradshaw Index (HBI), Simple Clinical Colitis Activity Index (SCCAI), and the Simple Endoscopic Score for Crohn's Disease (SES-CD).

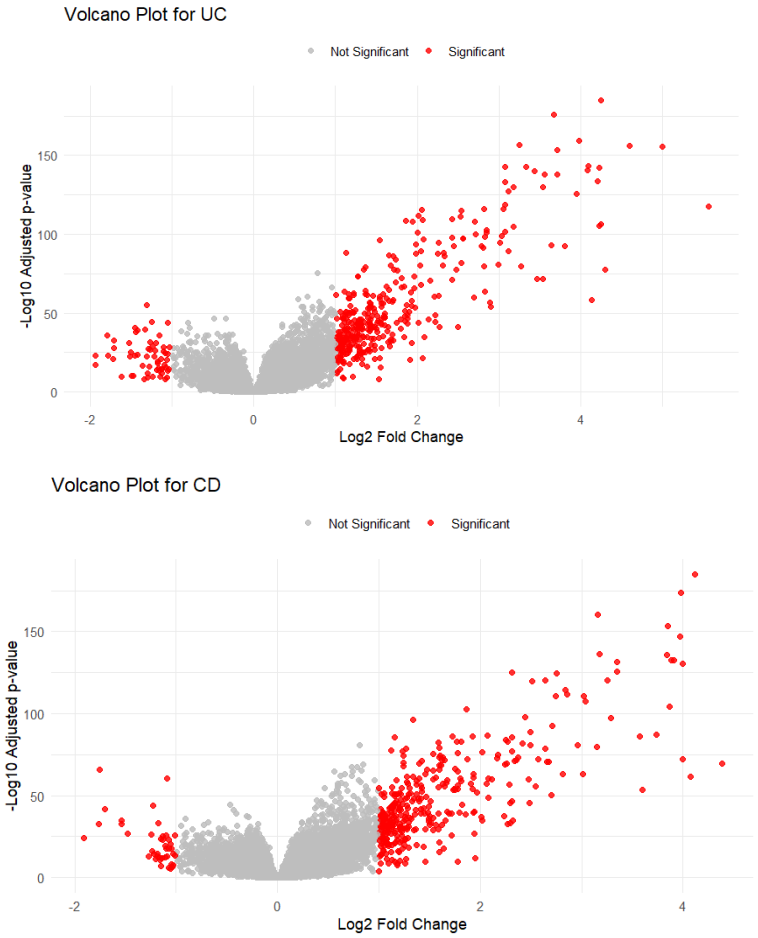
Data preprocessing

The data set consists of biopsy RNA-sequencing samples from adult patients with either Crohn's disease, ulcerative colitis, or healthy controls. There are 1151, 865, and 455 participants in each class, respectively. The sequencing was performed on whole blood from various abdominal tissues. Comprehensive metadata was included as well, such as sex, age, biopsy region, and if the tissue was inflamed at the time of sampling. In total, over 39,000 gene transcripts and 2471 patients were sequenced. The baseline accuracy for a model that predicted only the majority class in this dataset would be 47%.

Prior to training the machine learning models, preprocessing of the gene counts and metadata was essential. The dataset included over 39,000 genes, making it computationally impractical and overly complex for effective model training. To identify which genes would be most predictive, differentially expressed gene (DEG) analysis was performed to identify genes with significantly altered expression levels when compared to the healthy control samples.

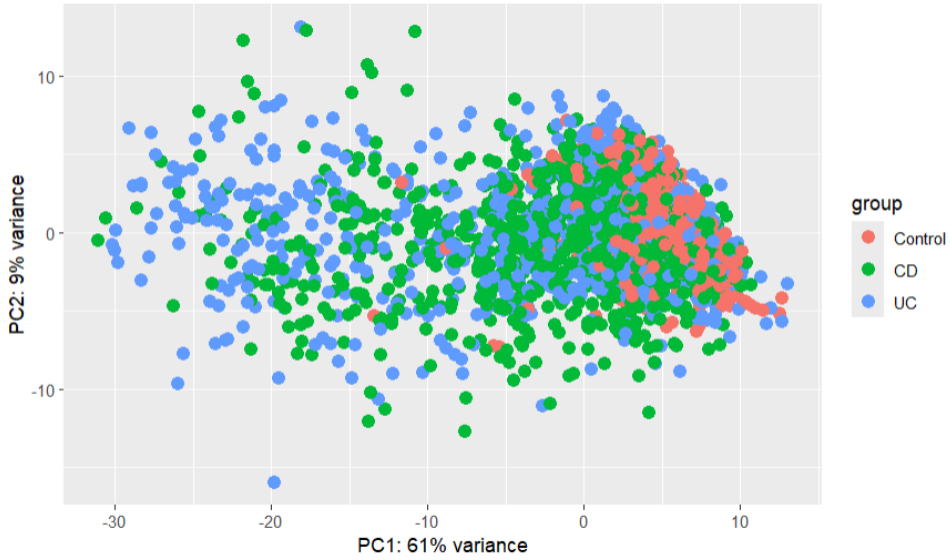
First, lowly expressed genes were removed. This was defined as any gene with less than 10 counts in at least 10% of participants. 23701 genes remained after this step. Next, the DESeq2 pipeline was used to identify DEGs. Figure 1 shows the volcano plots for UC and CD. They share similar profiles, in that the conditions appear to cause more upregulation than down regulation.

Figure 1. Volcano Plots for CD and UC



The adjusted p-values for each condition were then sorted into two separate dataframes, so that the most significant DEGs could be easily callable. To ensure that genes of varying expression levels can be compared, the variance stabilizing transformation function from the DESeq2 library was applied to each gene, followed by z-score normalization. Numerical metadata, such as age, was normalized as well. The PCA plot for the forty most significant genes is shown in figure 2. This plot shows the control samples clustered together, while the disease conditions have a larger spread among the two principal components that explain the most variance.

Figure 2. PCA plot of the top 40 most significant DEGs for UC and CD



XGBOOST

The subset of the data used for XGBoost consisted of thirty five genes, age, sex, biopsy region, and whether the tissue was inflamed or not. One hot encoding was used for sex, biopsy region, and inflamed (y/n). Eighteen of the most significant genes for UC and seventeen for CD were selected, ensuring any genes significant to both conditions were included. To maintain equal proportions, additional genes were alternately added from each condition's significance rankings.

Training-test splits were made at an 80-20% ratio. An xgb matrix was then made for the training and test data using the xgboost package in r. Parameters for the model were determined via the grid search method. Three values for the tested parameters were chosen to evaluate low, medium, and high settings. Parameters were evaluated using the first 500 participants from the training set to ease the computational load and 5-fold cross validation was used to evaluate accuracy of each parameter combination. The parameters checked and the result can be seen in Table 1.

Table 1. Parameter optimization for XGBoost

	Tested	Result
nrounds	2, 5, 10	10
eta	0.01, 0.3, 0.5	0.3
max_depth	3, 6, 9	9
gamma	0, 1, 5	1

Min_child_weight	1, 5, 10	5
subsample	0.5, 0.75, 1	1
colsample_bytree	0.5, 0.75, 1	1

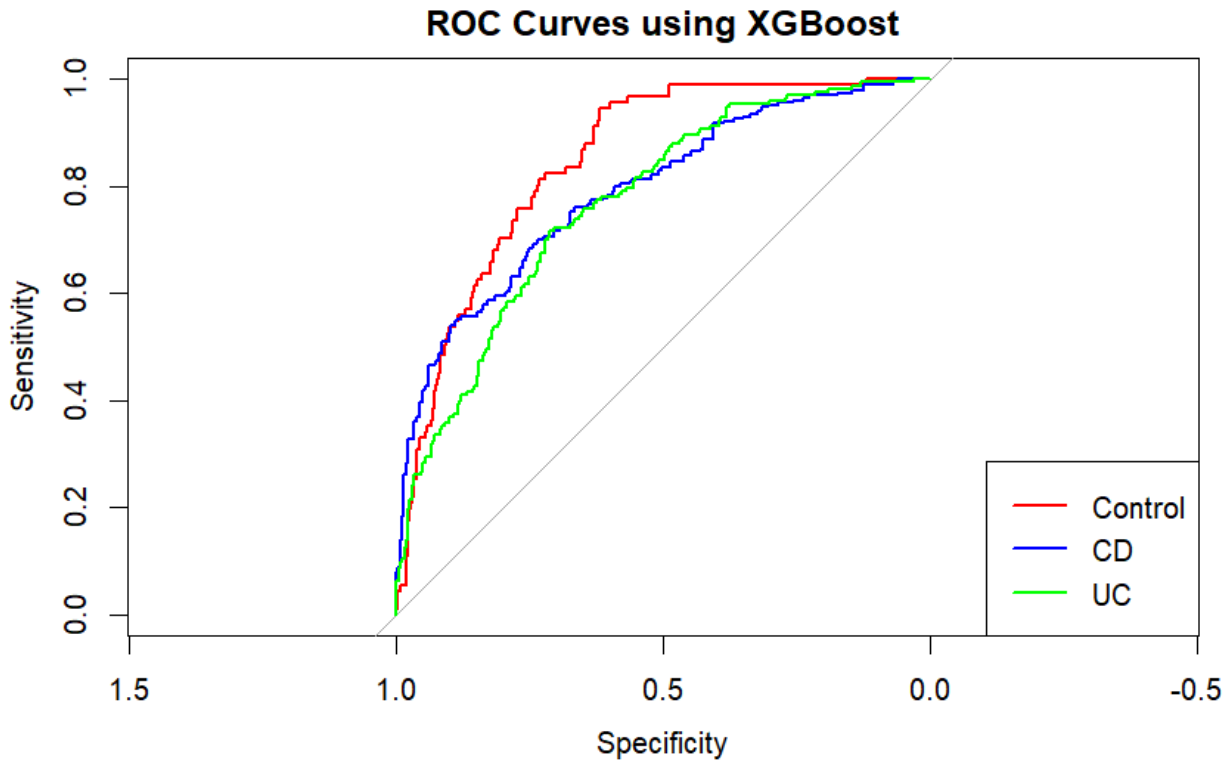
The model's performance was then evaluated using the test set. Class predictions were awarded to the class with the highest probability determined by the model. Accuracy, precision, recall, F1, and AUC were collected to evaluate the models ability to discriminate between the three classes, and are presented in Table 2 along with the other tested models.

Table 2. Model Evaluation

Model	Class	Precision	Recall	F1	AUC
XGBoost	Control	0.50	0.51	0.50	0.85
	CD	0.67	0.71	0.69	0.77
	UC	0.65	0.60	0.62	0.80
SVM	Control	1	1	1	1
	CD	0.75	0.81	0.78	0.88
	UC	0.74	0.66	0.70	0.88
Decision Tree	Control	0.97	0.75	0.72	1
	CD	0.69	0.75	0.72	0.79
	UC	0.67	0.6	0.63	0.79
Random Forest	Control	1	1	1	1
	CD	0.77	0.84	0.8	0.90
	UC	0.78	0.69	0.73	0.90
AdaBoost	Control	1	0.99	0.99	1.00
	CD	0.77	0.86	0.81	0.91
	UC	0.80	0.68	0.74	0.90

The ROC curve for each class based on a one vs all approach can be seen in Figure 3.

Figure 3. ROC Curves using XGBoost



The XGBoost model with optimized parameters and taking the class with the highest probability as the prediction achieved an overall accuracy of 61.5%. This is a significant improvement over the baseline accuracy of 47%. CD had the highest F1, followed by UC then Control. This indicates that the model was best at limiting both false positives and false negatives for CD. Interestingly, the AUC for control was higher than CD and UC, suggesting that the model is better at distinguishing healthy patients from diseased patients. A more conservative threshold could lead to a better performing model. Additionally, a high AUC for control indicates that a conservative model could be reasonably practical for diagnostic purposes, as the model has a good ability to tell healthy from disease tissues.

Support Vector Machine

A Support Vector Machine (SVM) model was then trained on 34 genes, age, sex, biopsy region, tissue inflammation status, log2 C-reactive protein, and disease activity according to HBI, SCCAI, and SES-CD. Initially, 3 data frames were assessed for most optimal performance, differing on the handling of NA's. For the first data frame, all 5 variables with any NA results were removed. In the second, only 2 of the variables were removed due to a high (43%) volume of NA's, and the remaining NA's were either replaced with mean (for continuous variables) or mode (for categorical). The third data frame did not remove any of the variables and only

replaced the NA values with the aforementioned criteria. As can be expected, the data frame which only removed the 2 variables with a large volume of NA's performed the best, and was used for the remaining analyses and models.

Categorical variables were factorized and continuous variables were normalized as previously stated. Seventeen of the most significant genes for UC and CD were selected based on lowest p-value and without repeating any genes that were significant for both diseases. Training-test splits were made at an 80-20% ratio. Various SVM metrics were tested to assess optimal performance based on linear versus radial kernels and several different C values. Each model was compared using 5-fold cross validation, the results of which are listed in Table 3.

Table 3. SVM Metrics Evaluation

Dataframe	SVM Combo	Mean CV Score
rem5: 38 Features	svcl_15: Linear, C=1.5	61.818181
rem2: 41 Features	svcl_05: Linear, C=0.5	0.765288513
rem2: 41 Features	svcl_1: Linear, C=1	0.762860181
rem2: 41 Features	svcl_15: Linear, C=1.5	0.765692553
rem2: 41 Features	svcr_05: Radial, C=0.5	0.633348873
rem2: 41 Features	svcr_1: Radial, C=1	0.664102564
rem2: 41 Features	svcr_15: Radial, C=1.5	0.687576984
noNA: 43 Features	svcl_05: Linear, C=0.5	0.765690099
noNA: 43 Features	svcl_1: Linear, C=1	0.765689281
noNA: 43 Features	svcl_15: Linear, C=1.5	0.764877929
noNA: 43 Features	svcr_05: Radial, C=0.5	0.630514865
noNA: 43 Features	svcr_1: Radial, C=1	0.665725269
noNA: 43 Features	svcr_15: Radial, C=1.5	0.684339754

The most successful SVM utilized a linear kernel and a moderate C regularization of 1.5. This was the model used to compare with the other classifiers and its efficiency is listed in Table 1. The fact that UC has a lower recall is expected, as the model is more conservative with predicting UC, but it is more often correct when it does. This is likely due to the overlapping nature of the diseases both involving inflammation of the GI tract, making it slightly more difficult to identify UC as opposed to a localized case of CD. The higher recall for CD shows that the model misses less true positives, presumably because CD will be more easily identified when it is aggressive. Figure 4 displays the ROC curve for this SVM, with an expectedly excellent performance for balancing the true negative rate and true positive rate for the healthy controls, and adequate performance for both diseases, with no clear distinction in the performance for either disease. These trends remain consistent through the remaining models, providing confidence in the consistency of the dataset.

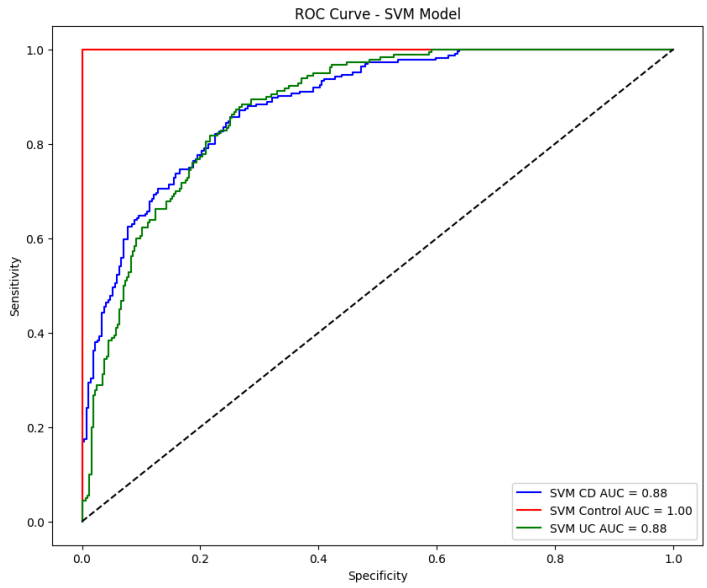


Figure 4. ROC Curve for SVM

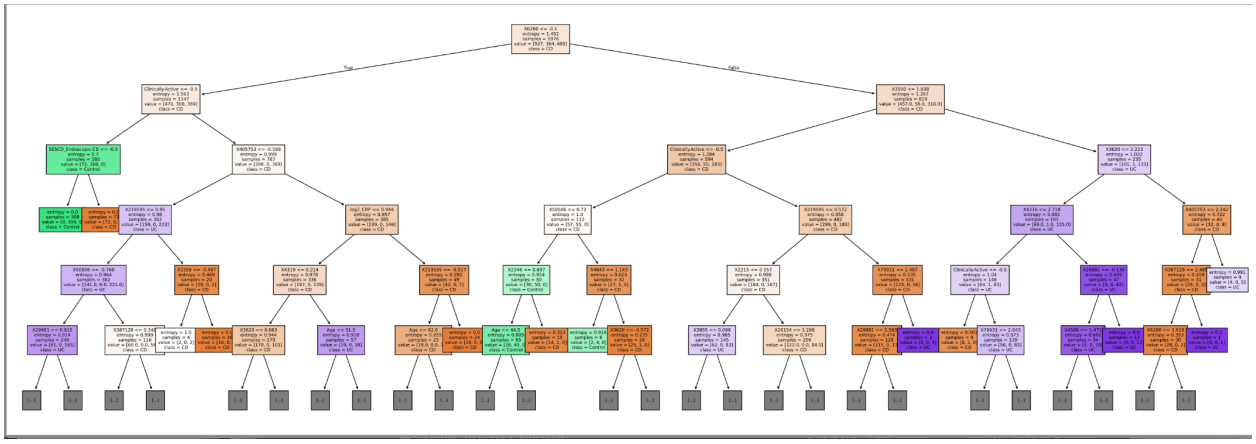
Decision Tree

The dataset that yielded the best performance in SVM was then used to train a Decision Tree model. For computational efficiency, RandomizedSearchCV was used with 50 iterations to check random combinations of specified parameters instead of every possible combination in order to determine the most optimal combination of hyper parameters. StratifiedKFold was utilized to ensure the folds were representative of the overall distribution of the data. The hyper parameters explored included gini and entropy impurity measures, max depths of 0 to 50 by increments of 10, square root and log2 methods of calculating the number of features per split, and several different values for minimum samples per leaf and per split. The best estimator's parameters are displayed in Figure 5. An output of the full tree revealed a depth of 19 and a clear level of complexity required of the model to differentiate the 2 diseases, as healthy controls are distinguished by a depth of seven. Only the depth up to 5 are shown in Figure 6. The ROC curve in Figure 6 displays a decrease in this model's AUC when compared to the SVM. This decrease is driven by lower precision and recall for all three classes.

Figure 5. Best Estimator for Decision Tree (DT)

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_features='sqrt',
                       min_samples_split=20, random_state=50)
```

Figure 6. Decision Tree



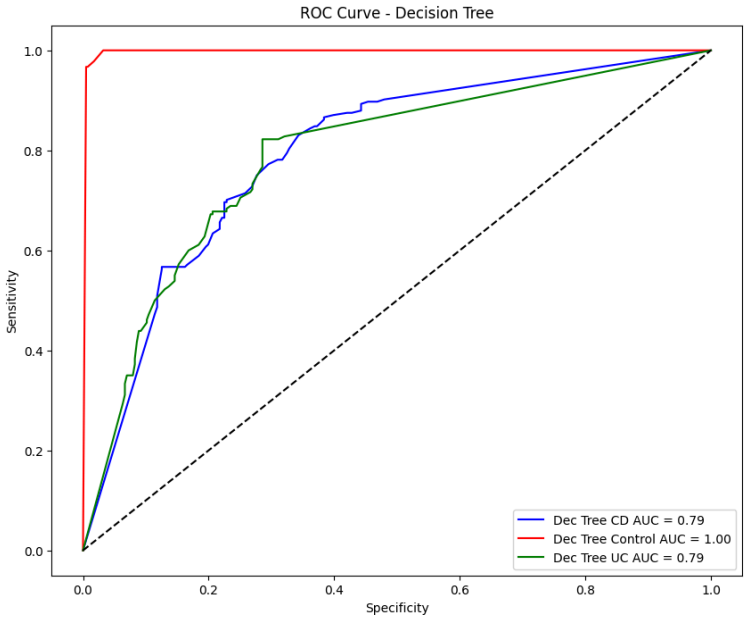


Figure 7. ROC for Best DT Estimator

Random Forest and AdaBoost

Lastly, the dataset was used to train a Random Forest model and then perform AdaBoosting to that model. RandomizedSearchCV was deployed once more to ascertain an optimal combination of hyper parameters, albeit not guaranteed to be the best, but it still performed well. The same combination of parameters were used as in the Decision Tree search and the best estimator is described in Figure 8. The AdaBoostClassifier was then applied with a varying number of estimators and learning rates to ascertain an optimal result, determined to be 100 estimators and a learning rate of 1.5. The AUC for both classifiers were very similar, with only a mild increase in classifying Crohn's disease with the AdaBoosted Classifier, as expected since Random Forest is already a strong learner. Figures 9 and 10 display the ROC curves for both classifiers.

Figure 8. Random Forest Best Estimator

```
RandomForestClassifier
RandomForestClassifier(max_depth=20, max_features='log2', min_samples_leaf=5,
                       oob_score=True, random_state=50)
```

Figure 9. ROC curve for Random Forest (RF)

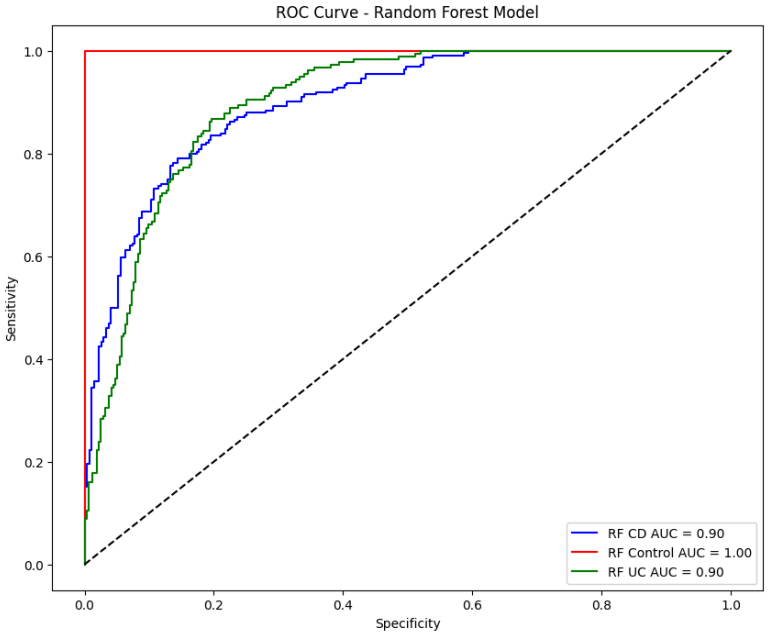
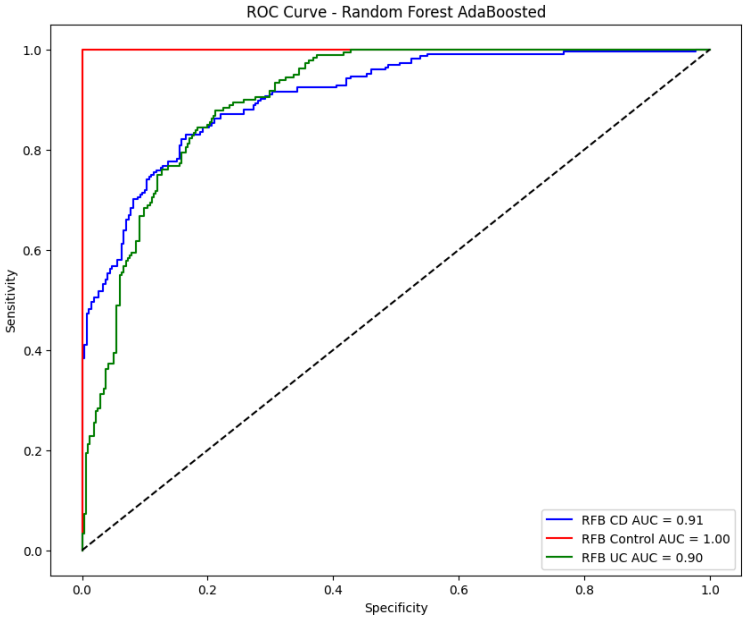


Figure 10. ROC curve for AdaBoosted RF



Discussion/Conclusion

This project demonstrates the potential of machine learning models to aid in the diagnosis of patients with suspected Crohn's Disease or Ulcerative Colitis. Among the models tested,

Adaboost achieved the highest overall accuracy with 82%. AdaBoosted Random Forest achieved the highest average AUC for each class, indicating that it was the best discriminator of the models tested. In general, the models were able to better discern control participants from those with either disease, as evident by their higher AUC. The PCA plot in figure 2 provides insight; control samples cluster together tightly along PC1, while CD and UC are spread more broadly. This indicates that healthy participants have distinct expression profiles in the top 40 most significant DEGs, providing strong predictive features for the models to base control predictions off of. UC and CD have similar profiles in the PCA plot and volcano plot, and share a number of top DEGs. This explains why the AUC for these classes is typically lower than control across the models. The decision tree classifier determined the feature with the most information gain would be expression of gene 6280 which corresponds to IL-6, a promoter of inflammation. This is a reasonable choice given the nature of the diseases, but also raises questions as to whether the model is simply classifying inflammation.

It is interesting that the SVM, Decision Tree, and Random Forest all had similar AUC's for classifying the 3 conditions, differing mostly in the minute details of precision and recall. This speaks to the complexity of the disease, as each algorithm approaches the data slightly differently and therefore picks up on the patterns from various angles. The AdaBoosted Random Forest likely performed the best because it can adapt better to the dataset's intricacy, where SVM may have struggled with non-linear patterns and Decision Tree may have been too easily influenced towards overfitting by that same complexity.

Discussion of Limitations and Alternative Interpretations

One limitation of this project was the number of participants compared to the number of DEGs. The number of significant DEGs was far greater than the number that could reasonably be included given the number of participants. Furthermore, the classes for each disease were imbalanced, placing limitations on the number of features that could be included and adding complexity to the training process. Including more participants in the study would allow for a broader investigation of the transcriptome.

A potential complication if this model were to be implemented in a diagnostic setting, is that the scope of this project was narrow. There are many more conditions that impact the gut transcriptome, and the ability of these models to differentiate between them is unknown. Furthermore, the primary symptoms of UC and CD are inflammation, which can arise for many reasons other than these diseases. It is reasonable to assume that these models may attribute general inflammation to disease status, which is not always true.

Clear Statement of Future Work

Expanding the data set to include more participants and more gastrointestinal conditions is necessary to evaluate the real world application of a machine learning classifier in aiding diagnostic decision making. Furthermore, the thresholds for each model will need to be fine tuned for the specific goals of their application. For example, if a practice has other methods for evaluating CD and UC, a threshold that is more strict or relaxed may be appropriate to control the false positive rate.

References

1. Mo A, Krishnakumar C, Arafat D, Dhere T, Iskandar H, Dodd A, Prince J, Kugathasan S, Gibson G. African Ancestry Proportion Influences Ileal Gene Expression in Inflammatory Bowel Disease. Cell Mol Gastroenterol Hepatol. 2020;10(1):203-205. doi: 10.1016/j.jcmgh.2020.02.001. Epub 2020 Feb 10. PMID: 32058087; PMCID: PMC7296223
2. Wiesen, Ari MD*; Katz, Seymour MD, MACG, FACP; Liu, Blanche Fung MD; Oustecky, David MD; Sommers, Camille MD; Krohn, Natan MD. Factors Associated with Conversion of an Ulcerative Colitis Diagnosis to Crohn's Disease: 1075. American Journal of Gastroenterology 103():p S420, September 2008.

Data Aquisition and preprocessing

```
%%R
library(GEOquery)
library(DESeq2)
library(dplyr)

# Load in data
gse = getGEO("GSE193677")

# Get metadata
metaData = pData(gse[[1]])

# Get disease status
class = substring(metaData$characteristics_ch1.4,14)

# Get Expression Data
expr_data = read.table("GSE193677_raw_counts_GRCh38.p13_NCBI.tsv",header=T)

# Make rownames Gene IDs
rownames(expr_data) = expr_data$GeneID

# remove Gene ID column
expr_data$GeneID = NULL

# find samples that are in metadata but not in the expression data
missing = setdiff(rownames(metaData),colnames(expr_data))

# find the indexes of these samples in the metadata
missing_id = which(rownames(metaData) %in% missing)

# remove from class
class_cln = data.frame(Class = class[-missing_id])
rownames(class_cln) = colnames(expr_data)
```

Show hidden output

DEG identification

```
%%R
library(DESeq2)
# Define class levels
class_cln$Class = factor(class_cln$Class, levels = c("Control", "CD", "UC"))

# Remove lowly expressed genes
# Set minimum counts
min_counts = 10

# Define minimum number of samples with minimum number of counts. 10% of total
min_samples = ceiling(0.1 * ncol(expr_data))

# Take the rows that have at least 10 counts in 10 percent of samples
keep = which(rowSums(expr_data > min_counts) >  min_samples)

# Filter data
expr_data_filt = expr_data[keep,]

# Make DESeq dataset
dds = DESeqDataSetFromMatrix(countData = expr_data_filt,
                             colData = class_cln,
                             design= ~ Class)
# Find DEGS
dds = DESeq(dds)

# Get the results
res_CD = results(dds, contrast = c("Class", "CD", "Control"))
res_UC = results(dds, contrast = c("Class", "UC", "Control"))
res_UC_vs_CD = results(dds, contrast = c("Class","CD","UC"))
# Find signifcant DEGS
sig_CD = subset(res_CD, padj < 0.05 & abs(log2FoldChange) > 1)
sig_UC = subset(res_UC, padj < 0.05 & abs(log2FoldChange) > 1)
```

```
# Quality check and visualizations
library(DESeq2)
plotMA(res_CD)
plotMA(res_UC)
plotDispEsts(dds)
hist(results(dds)$pvalue, breaks=50)

# Volcano plot

# Convert to dataframe
res_UC_df = as.data.frame(res_UC)

# Add significance column
res_UC_df$Significance = "Not Significant"
# Define which genes are significant
res_UC_df$Significance[res_UC_df$padj < 0.05 & abs(res_UC_df$log2FoldChange) > 1] = "Significant"

library(ggplot2)

# Volcano plot for UC
ggplot(res_UC_df, aes(x=log2FoldChange, y=-log10(padj), color=Significance)) +
  geom_point(alpha=0.8, size=1.5) +
  scale_color_manual(values=c("gray", "red")) +
  theme_minimal() +
  labs(
    title = "Volcano Plot for CD",
    x = "Log2 Fold Change",
    y = "-Log10 Adjusted p-value"
  ) +
  theme(
    legend.title = element_blank(),
    legend.position = "top"
  )

# Volcano plot for CD

res_CD_df = as.data.frame(res_CD)
res_CD_df$Significance = "Not Significant"
res_CD_df$Significance[res_CD_df$padj < 0.05 & abs(res_CD_df$log2FoldChange) > 1] = "Significant"

ggplot(res_CD_df, aes(x=log2FoldChange, y=-log10(padj), color=Significance)) +
  geom_point(alpha=0.8, size=1.5) +
  scale_color_manual(values=c("gray", "red")) +
  theme_minimal() +
  labs(
    title = "Volcano Plot for CD",
    x = "Log2 Fold Change",
    y = "-Log10 Adjusted p-value"
  ) +
  theme(
    legend.title = element_blank(),
    legend.position = "top"
  )

# Plot PCA
# filter for significant DEGs
res = results(dds)
# Take top 40 genes
top_genes = rownames(res[order(res$padj), ][1:40, ])

# Perform VST
vst_data = varianceStabilizingTransformation(dds, blind = TRUE)
vst_subset = vst_data[top_genes,]

# Plot PCA
plotPCA(vst_subset, intgroup = "Class")
```

Show hidden output

Next steps: Explain error

Metadata Analysis

```
%%R
# selected metadata
final.meta <- metaData[,c("ibd_disease:ch1",
  "demographics_gender:ch1",
  "study_eligibility_age_at_endo:ch1",
  "regionre:ch1",
  "typere:ch1",
  "crp_jjmgl_log2:ch1",
  "log2_fecalcalpro_mgperg:ch1",
  "ibd_clinicianmeasure_inactive_active:ch1",
  "ibd_endoseverity_4levels:ch1",
  "ibdmesuc_mayo_score:ch1",
  "max_nancy:ch1",
  "max_ghas_sum7:ch1")]

final.meta <- final.meta %>% rename(
 "Disease.type" = "ibd_disease:ch1", # UC, CD, Control
 "Sex" = "demographics_gender:ch1",
 "Age" = "study_eligibility_age_at_endo:ch1",
 "Tissue.region" = "regionre:ch1",
 "Inflamed.yn" = "typere:ch1", # NonI, I
 "log2_CRP" = "crp_jjmgl_log2:ch1", # C-reactive protein, IBD activity biomarker
 "log2_Calprotectin" = "log2_fecalcalpro_mgperg:ch1", # IBD activity biomarker
 "Clinically.Active" = "ibd_clinicianmeasure_inactive_active:ch1", # inactive disease: HBI < 5 or SCCAI < 5, active disease: HBI >7 or SSCAI

 ### unsure if we should keep it
 "SESCD_Endoscopic.CD" = "ibd_endoseverity_4levels:ch1", # for Crohn's, Simple Endoscopic Score for Crohn’s Disease (SES-CD) - inactive (0–2
 "Mayo_Endoscopic.UC" = "ibdmesuc_mayo_score:ch1", # endoscopic measure for UC: normal/inactive disease (0); mild disease (1); moderate dise
 "Max.Nancy_Histo.UC" = "max_nancy:ch1", # control and UC, histological scoring of disease activity
 "Max.GHAS_Histo.CD"  = "max_ghas_sum7:ch1", # control and CD,  General histology activity score, scoring disease activity
)

# Remove samples with no expression data
final.meta = final.meta[!(rownames(final.meta) %in% missing),]
```

Create final data frame for machine learning

```
%%R
# Sort genes by adjusted p-value

pval_CD_sort = res_CD_df[order(res_CD_df$padj),]
pval_UC_sort = res_UC_df[order(res_UC_df$padj),]

# Take 200 of the most significant DEGS for each class
top_CD = rownames(pval_CD_sort)[1:200]
top_UC = rownames(pval_UC_sort)[1:200]

# Find the top unique genes
top_genes = unique(c(top_CD,top_UC))

# Pull these out of the expression data
top_expr = as.data.frame(t(expr_data[top_genes,]))

#Combine the metadata and expression data
model_data = cbind(final.meta, top_expr)
```

SVM

```
import numpy as np
import pandas as pd
from sklearn import svm
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import graphviz
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.tree import plot_tree
from sklearn.svm import SVC, LinearSVC  # Import specific classifiers
import pickle
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.preprocessing import label_binarize


# Load the objects from the pickle file
with open('/content/241208_workspace.pkl', 'rb') as f: # or the pathname to the pickle
    loaded_objects = pickle.load(f)

## Accessing the objects from the loaded workspace
# random_search_rf_loaded = loaded_objects['random_search_rf']
# X_rem2_loaded = loaded_objects['X_rem2']
# y_rem2_loaded = loaded_objects['y_rem2']


# Read the TSV file
model_data = pd.read_csv("/model_data.tsv", sep='\t')


model_data.head() # this data is not normalized
```

	Disease.type	Sex	Age	Tissue.region	Inflamed.yn	log2_CRP	log2_Calprotectin	Clinically.Active	SESCD_Endoscopic.CD
GSM5976499	UC	Male	44	Rectum	NonI	-1.733045	NaN	Inactive	Inactive
GSM5976500	UC	Male	44	LeftColon	NonI	-1.733045	NaN	Inactive	Inactive
GSM5976501	UC	Male	44	Ileum	NonI	-1.733045	NaN	Inactive	Inactive
GSM5976502	CD	Female	60	RightColon	NonI	1.436490	4.05398	Active	Inactive
GSM5976503	CD	Female	60	LeftColon	NonI	1.436490	4.05398	Active	Inactive

5 rows × 312 columns

```
%%R
# we have 11 features: check how many have >50% !is.na
norm.data <- read.csv("ModelDataNormalized.tsv", sep="\t", header=T)
sum(is.na(norm.data[,c(14:ncol(norm.data))])) # no expression data missing
sapply(norm.data[,c(2:13)], function(df){
  paste0(sum(is.na(df)), " (",
         round(sum(is.na(df))/length(df)*100, 2), "%)", sep="")
})

# 2 cols with 65% and 78% missing so remove: 8, 11 (calprotectin and UC endoscopic)
# 2 cols with 43.8% missing: 12:13 (histologic scores)
# try a model without these 2 cols and a model with replacing NA's with median
norm.data.rem2 <- norm.data[,-c(8,11:13)]
norm.data.noNA <- norm.data[,-c(8,11)] %>% mutate(
  Max.Nancy_Histo.UC = case_when(
    is.na(Max.Nancy_Histo.UC) ~ median(norm.data$Max.Nancy_Histo.UC,
                                       na.rm = TRUE),
    TRUE ~ Max.Nancy_Histo.UC),
    Max.GHAS_Histo.CD = case_when(
      is.na(Max.GHAS_Histo.CD) ~ median(norm.data$Max.GHAS_Histo.CD,
                                        na.rm = TRUE),
      TRUE ~ Max.GHAS_Histo.CD
    )
)

# now we have 7-9 features
# find 34 total DEGs (17 from each without duplicates)
cd.deg <- read.csv("CD_DEGS_sorted_pval.txt", header = TRUE, sep = "\t")
uc.deg <- read.csv("UC_DEGS_sorted_pval.txt", header = TRUE, sep = "\t")

cd.deg <- cd.deg[order(cd.deg$padj),]
uc.deg <- uc.deg[order(uc.deg$padj),]


find.34 <- data.frame(
  GeneID = gsub("^X", "", colnames(norm.data.rem2[,c(10:ncol(norm.data.rem2))])),
```

```
  cd.deg.padj = NA,
  uc.deg.padj = NA
)

find.34$cd.deg.padj <- cd.deg$padj[match(find.34$GeneID, cd.deg$GeneID)]
find.34$uc.deg.padj <- uc.deg$padj[match(find.34$GeneID, uc.deg$GeneID)]
# write.csv(find.34, "241208_Match.Colnames.csv")

cd.deg.top20 <- head(find.34[order(find.34$cd.deg.padj),],20)
uc.deg.top20 <- head(find.34[order(find.34$uc.deg.padj),],20)

# checks if two sets contain the same elements, disregarding their order
setequal(cd.deg.top20$GeneID, uc.deg.top20$GeneID)
# which ones are not equal - Warning: 20 mismatches does not mean all are not equal
all.equal(cd.deg.top20$GeneID, uc.deg.top20$GeneID)

# all 20 are not the same so I can use the top 17 from each

cd.deg.top17 <- head(find.34[order(find.34$cd.deg.padj),],17)

uc.deg.find17 <- find.34[order(find.34$uc.deg.padj),]
uc.deg.find17 <- uc.deg.find17[!(uc.deg.find17$GeneID %in% cd.deg.top17$GeneID),]
uc.deg.top17 <- head(uc.deg.find17, 17)

keep.34 <- rbind(cd.deg.top17, uc.deg.top17)
# Add 'X' in front of each GeneID to match back
keep.34$GeneID <- paste0("X", keep.34$GeneID)


# filter model data for just the 34 genes
norm.data.rem2.cut <- norm.data.rem2[,c(1:9,
                                        which(colnames(norm.data.rem2) %in%
                                                keep.34$GeneID))]


norm.data.noNA.cut <- norm.data.noNA[,c(1:11,
                                        which(colnames(norm.data.noNA) %in%
                                                keep.34$GeneID))]

# write.csv(norm.data.rem2.cut, "241208_Norm.data.Rem2.csv")
# write.csv(norm.data.noNA.cut, "241208_Norm.data.NoNA.csv")


# now we have 41-43 features:
## norm_rem2 has 41 features because we removed 2 with 43% NA
## norm_noNA has 43 features because we kept the 2 cols and replaced NA's with the col's median


norm_rem2 = pd.read_csv("/content/241208_Norm.data.Rem2.csv")
# use set_index to make the ID column my index for the df
norm_rem2 = norm_rem2.set_index('ID')
norm_rem2.head()
```

ID	Disease.type	Sex	Age	Tissue.region	Inflamed.yn	log2_CRP	Clinically.Active	SESCD_Endoscopic.CD	X2346	X1116
GSM5976499	UC	Male	44	Rectum	NonI	-1.733045	Inactive	Inactive	-0.734807	-0.239503
GSM5976500	UC	Male	44	LeftColon	NonI	-1.733045	Inactive	Inactive	-0.798520	-0.661091
GSM5976501	UC	Male	44	Ileum	NonI	-1.733045	Inactive	Inactive	0.220419	0.206250
GSM5976502	CD	Female	60	RightColon	NonI	1.436490	Active	Inactive	0.617282	-0.735794
GSM5976503	CD	Female	60	LeftColon	NonI	1.436490	Active	Inactive	-0.618784	-0.662444

5 rows × 42 columns

```
norm_noNA = pd.read_csv("/content/241208_Norm.data.NoNA.csv")
norm_noNA = norm_noNA.set_index('ID')
norm_noNA.head()
```

ID	Disease.type	Sex	Age	Tissue.region	Inflamed.yn	log2_CRP	Clinically.Active	SESCD_Endoscopic.CD	Max.Nancy_Histo.UC
GSM5976499	UC	Male	44	Rectum	NonI	-1.733045	Inactive	Inactive	0
GSM5976500	UC	Male	44	LeftColon	NonI	-1.733045	Inactive	Inactive	0
GSM5976501	UC	Male	44	Ileum	NonI	-1.733045	Inactive	Inactive	0
GSM5976502	CD	Female	60	RightColon	NonI	1.436490	Active	Inactive	0
GSM5976503	CD	Female	60	LeftColon	NonI	1.436490	Active	Inactive	0

5 rows × 44 columns

```
# set my X and y for both sets, one-hot encoding the y's

# The algorithm requires the variables to be coded into its equivalent integer codes
# We can convert the string categorical values into an integer code using numpy
# setting 2 objects: vals and y2
# where vals = np.unique(y['diagnosis'].values - an array of the unique values in Diagnosis col
# and y = an inverse array of the col - where 0,1 values are attributed to the
## values in the col based on the unique values, and y is a map of the original col but in 0,1's
vals_rem2, y_rem2 = np.unique(norm_rem2['Disease.type'].values, return_inverse=True)
vals_rem2
```

```
array(['CD', 'Control', 'UC'], dtype=object)
```

```
y_rem2 # 0: CD, 1: Control, 2: UC
```

```
array([2, 2, 2, ..., 1, 1, 1])
```

```
# save my hc_cf_top10 df as a new object called X so I can store the gene expression values
X_rem2 = norm_rem2
# use pop to remove the Y column and save only the gene expression values as my X df
X_rem2.pop('Disease.type')
X_rem2.head()
```

ID	Sex	Age	Tissue.region	Inflamed.yn	log2_CRP	Clinically.Active	SESCD_Endoscopic.CD	X2346	X1116	X2215	..
GSM5976499	Male	44	Rectum	NonI	-1.733045	Inactive	Inactive	-0.734807	-0.239503	-0.795064	
GSM5976500	Male	44	LeftColon	NonI	-1.733045	Inactive	Inactive	-0.798520	-0.661091	-1.133618	
GSM5976501	Male	44	Ileum	NonI	-1.733045	Inactive	Inactive	0.220419	0.206250	-0.501224	
GSM5976502	Female	60	RightColon	NonI	1.436490	Active	Inactive	0.617282	-0.735794	-0.615965	
GSM5976503	Female	60	LeftColon	NonI	1.436490	Active	Inactive	-0.618784	-0.662444	-0.820484	

5 rows × 41 columns

```
vals_noNA, y_noNA = np.unique(norm_noNA['Disease.type'].values, return_inverse=True)
vals_noNA
```

```
array(['CD', 'Control', 'UC'], dtype=object)
```

```
y_noNA
```

```
array([2, 2, 2, ..., 1, 1, 1])
```

```
X_noNA = norm_noNA
X_noNA.pop('Disease.type')
X_noNA.head()
```

ID	Sex	Age	Tissue.region	Inflamed.yn	log2_CRP	Clinically.Active	SESCD_Endoscopic.CD	Max.Nancy_Histo.UC	Max.GHAS_His
GSM5976499	Male	44	Rectum	NonI	-1.733045	Inactive	Inactive	0	
GSM5976500	Male	44	LeftColon	NonI	-1.733045	Inactive	Inactive	0	
GSM5976501	Male	44	Ileum	NonI	-1.733045	Inactive	Inactive	0	
GSM5976502	Female	60	RightColon	NonI	1.436490	Active	Inactive	0	
GSM5976503	Female	60	LeftColon	NonI	1.436490	Active	Inactive	0	

5 rows × 43 columns

```
print(X_noNA.columns)
```

```
Index(['Sex', 'Age', 'Tissue.region', 'Inflamed.yn', 'log2_CRP',
       'Clinically.Active', 'SESCD_Endoscopic.CD', 'Max.Nancy_Histo.UC',
       'Max.GHAS_Histo.CD', 'X2346', 'X1116', 'X2215', 'X29881', 'X6289',
       'X6279', 'X219595', 'X6288', 'X366', 'X2358', 'X100528017', 'X405753',
       'X4843', 'X3620', 'X3772', 'X2357', 'X6280', 'X90527', 'X4314', 'X6947',
       'X50506', 'X387129', 'X79931', 'X3505', 'X4586', 'X4319', 'X3502',
       'X4316', 'X3855', 'X3500', 'X26154', 'X8710', 'X5655', 'X28454'],
      dtype='object')
```

```
print(X_rem2.columns)
```

```
Index(['Sex', 'Age', 'Tissue.region', 'Inflamed.yn', 'log2_CRP',
       'Clinically.Active', 'SESCD_Endoscopic.CD', 'X2346', 'X1116', 'X2215',
       'X29881', 'X6289', 'X6279', 'X219595', 'X6288', 'X366', 'X2358',
       'X100528017', 'X405753', 'X4843', 'X3620', 'X3772', 'X2357', 'X6280',
       'X90527', 'X4314', 'X6947', 'X50506', 'X387129', 'X79931', 'X3505',
       'X4586', 'X4319', 'X3502', 'X4316', 'X3855', 'X3500', 'X26154', 'X8710',
       'X5655', 'X28454'],
      dtype='object')
```

```
# need to factorize all categorical variables to integer equivalents
cols_to_factorize = [0, 2, 3, 5, 6]

# Dictionary to store the mappings for each column
category_mapping = {}

# Factorize each of the specified columns
for idx in cols_to_factorize:
    col_name = X_rem2.columns[idx]  # Get the actual column name by index
    # Factorize the column to get the numerical labels and the unique categories
    new_col, labels = pd.factorize(X_rem2[col_name])

    # Replace the original column with the numerical labels
    X_rem2[col_name] = new_col

    # Store the mapping (dictionary) of numerical labels to original categories
    category_mapping[col_name] = dict(enumerate(labels))

# Show the modified DataFrame and the mappings for each column
print(X_rem2.head())
print("\nCategory Mappings:", category_mapping)
```

```
             Sex  Age  Tissue.region  Inflamed.yn  log2_CRP  Clinically.Active  \
ID
GSM5976499    0   44              0            0 -1.733045                  0
GSM5976500    0   44              1            0 -1.733045                  0
GSM5976501    0   44              2            0 -1.733045                  0
GSM5976502    1   60              3            0  1.436490                  1
GSM5976503    1   60              1            0  1.436490                  1

             SESCD_Endoscopic.CD     X2346     X1116     X2215  ...     X4586  \
ID                                                              ...
GSM5976499                    0 -0.734807 -0.239503 -0.795064  ...  0.816403
GSM5976500                    0 -0.798520 -0.661091 -1.133618  ... -0.475171
GSM5976501                    0  0.220419  0.206250 -0.501224  ... -0.721310
GSM5976502                    0  0.617282 -0.735794 -0.615965  ... -0.723736
GSM5976503                    0 -0.618784 -0.662444 -0.820484  ... -0.684677

                X4319     X3502     X4316     X3855     X3500    X26154  \
ID
```

```
GSM5976499 -0.609817  0.387815 -0.110065 -0.081105  0.785482  0.696046
GSM5976500 -1.190500 -0.819715 -0.942298 -0.273055 -0.933020 -0.671092
GSM5976501 -1.190500  1.236697  0.505472 -0.824398  1.434299 -0.552254
GSM5976502 -0.327821 -0.864031 -0.698956 -0.641507 -0.468309 -0.812577
GSM5976503 -0.160902 -0.945777 -0.493108 -0.885334 -0.547335 -0.665613

               X8710     X5655    X28454
ID
GSM5976499  1.513917 -0.102172  0.265163
GSM5976500 -0.605820 -0.151610 -0.938057
GSM5976501 -0.605820 -0.394379  0.501208
GSM5976502 -0.605820 -0.497456 -0.623298
GSM5976503 -0.605820  0.009752 -0.999135

[5 rows x 41 columns]

Category Mappings: {'Sex': {0: 'Male', 1: 'Female'}, 'Tissue.region': {0: 'Rectum', 1: 'LeftColon', 2: 'Ileum', 3: 'RightColon', 4: 'Tra
```

```
X_rem2[X_rem2.isna().any(axis=1)]
```

ID	Sex	Age	Tissue.region	Inflamed.yn	log2_CRP	Clinically.Active	SESCD_Endoscopic.CD	X2346	X1116	X2215	...
GSM5976515	1	32	2	0	NaN	-1	0	1.157925	-0.815981	-0.441058	...
GSM5976516	1	32	0	0	NaN	-1	0	-0.571475	-1.010013	-0.608151	...
GSM5976596	1	46	0	0	NaN	0	2	3.645918	-0.545475	0.466389	...
GSM5976607	1	54	0	1	NaN	0	3	0.006387	1.827820	2.406125	...
GSM5976608	1	54	6	0	NaN	0	3	-0.521623	-0.647447	-0.580656	...
...	...	...	...	...	...	...	...	...	...	...	...
GSM5978853	1	21	0	1	NaN	0	1	-0.222561	1.986696	1.954445	...
GSM5978854	1	21	2	0	NaN	0	1	3.269520	-0.298827	0.626795	...
GSM5978918	1	71	0	0	NaN	-1	-1	-0.644612	-0.688761	-0.755336	...
GSM5978929	1	61	6	0	NaN	-1	-1	-0.686691	-0.418322	-0.791771	...
GSM5978930	1	61	0	0	NaN	-1	-1	0.071240	-0.484228	-0.150165	...

89 rows × 41 columns

```
rows_with_na_indices = X_rem2[X_rem2.isna().any(axis=1)].index


category_mapping_noNA = {}

# Factorize each of the specified columns
for idx in cols_to_factorize:
    col_name = X_noNA.columns[idx]  # Get the actual column name by index
    # Factorize the column to get the numerical labels and the unique categories
    new_col, labels = pd.factorize(X_noNA[col_name])

    # Replace the original column with the numerical labels
    X_noNA[col_name] = new_col

    # Store the mapping (dictionary) of numerical labels to original categories
    category_mapping_noNA[col_name] = dict(enumerate(labels))

# Show the modified DataFrame and the mappings for each column
print(X_noNA.head())
print("\nCategory Mappings:", category_mapping_noNA)
```

```
            Sex  Age  Tissue.region  Inflamed.yn  log2_CRP  Clinically.Active  \
ID
GSM5976499    0   44              0           0 -1.733045                  0
GSM5976500    0   44              1           0 -1.733045                  0
GSM5976501    0   44              2           0 -1.733045                  0
GSM5976502    1   60              3           0  1.436490                  1
GSM5976503    1   60              1           0  1.436490                  1

            SESCD_Endoscopic.CD  Max.Nancy_Histo.UC  Max.GHAS_Histo.CD  \
ID
GSM5976499                    0                   0                  2
```

```
GSM5976500                   0                  0                  2
GSM5976501                   0                  0                  2
GSM5976502                   0                  0                  1
GSM5976503                   0                  0                  1

                X2346  ...     X4586     X4319     X3502     X4316     X3855  \
ID                     ...
GSM5976499 -0.734807  ...  0.816403 -0.609817  0.387815 -0.110065 -0.081105
GSM5976500 -0.798520  ... -0.475171 -1.190500 -0.819715 -0.942298 -0.273055
GSM5976501  0.220419  ... -0.721310 -1.190500  1.236697  0.505472 -0.824398
GSM5976502  0.617282  ... -0.723736 -0.327821 -0.864031 -0.698956 -0.641507
GSM5976503 -0.618784  ... -0.684677 -0.160902 -0.945777 -0.493108 -0.885334

                X3500    X26154     X8710     X5655    X28454
ID
GSM5976499  0.785482  0.696046  1.513917 -0.102172  0.265163
GSM5976500 -0.933020 -0.671092 -0.605820 -0.151610 -0.938057
GSM5976501  1.434299 -0.552254 -0.605820 -0.394379  0.501208
GSM5976502 -0.468309 -0.812577 -0.605820 -0.497456 -0.623298
GSM5976503 -0.547335 -0.665613 -0.605820  0.009752 -0.999135

[5 rows x 43 columns]

Category Mappings: {'Sex': {0: 'Male', 1: 'Female'}, 'Tissue.region': {0: 'Rectum', 1: 'LeftColon', 2: 'Ileum', 3: 'RightColon', 4: 'Tra
```

```
# Function to replace NaN with median (for continuous columns) or mode (for categorical columns)
def my_replace_na(df):
    for col in df.columns:
        if df[col].dtype in [np.float64, np.int64]:  # Continuous column (numeric)
            # Directly modify the column using df.loc
            df.loc[:, col] = df[col].fillna(df[col].median())  # Avoid chained assignment
        elif df[col].dtype == 'object':  # Categorical column (string or object type)
            mode_val = df[col].mode()[0]  # Mode returns a Series, get the first value
            # Directly modify the column using df.loc
            df.loc[:, col] = df[col].fillna(mode_val)  # Avoid chained assignment
    return df

# Apply the function to replace NaNs
X_rem2 = my_replace_na(X_rem2)
X_rem2.head()
```

ID	Sex	Age	Tissue.region	Inflamed.yn	log2_CRP	Clinically.Active	SESCD_Endoscopic.CD	X2346	X1116	X2215	...
GSM5976499	0	44	0	0	-1.733045	0	0	-0.734807	-0.239503	-0.795064	...
GSM5976500	0	44	1	0	-1.733045	0	0	-0.798520	-0.661091	-1.133618	...
GSM5976501	0	44	2	0	-1.733045	0	0	0.220419	0.206250	-0.501224	...
GSM5976502	1	60	3	0	1.436490	1	0	0.617282	-0.735794	-0.615965	...
GSM5976503	1	60	1	0	1.436490	1	0	-0.618784	-0.662444	-0.820484	...

5 rows × 41 columns

```
X_rem2.loc[rows_with_na_indices]
```

ID	Sex	Age	Tissue.region	Inflamed.yn	log2_CRP	Clinically.Active	SESCD_Endoscopic.CD	X2346	X1116	X2215	...
GSM5976515	1	32	2	0	1.930563	-1	0	1.157925	-0.815981	-0.441058	...
GSM5976516	1	32	0	0	1.930563	-1	0	-0.571475	-1.010013	-0.608151	...
GSM5976596	1	46	0	0	1.930563	0	2	3.645918	-0.545475	0.466389	...
GSM5976607	1	54	0	1	1.930563	0	3	0.006387	1.827820	2.406125	...
GSM5976608	1	54	6	0	1.930563	0	3	-0.521623	-0.647447	-0.580656	...
...	...	...	...	...	...	...	...	...	...	...	...
GSM5978853	1	21	0	1	1.930563	0	1	-0.222561	1.986696	1.954445	...
GSM5978854	1	21	2	0	1.930563	0	1	3.269520	-0.298827	0.626795	...
GSM5978918	1	71	0	0	1.930563	-1	-1	-0.644612	-0.688761	-0.755336	...
GSM5978929	1	61	6	0	1.930563	-1	-1	-0.686691	-0.418322	-0.791771	...
GSM5978930	1	61	0	0	1.930563	-1	-1	0.071240	-0.484228	-0.150165	...

89 rows × 41 columns

```
X_noNA[X_noNA.isna().any(axis=1)]
```

ID	Sex	Age	Tissue.region	Inflamed.yn	log2_CRP	Clinically.Active	SESCD_Endoscopic.CD	Max.Nancy_Histo.UC	Max.GHAS_Histo.
GSM5976515	1	32	2	0	NaN	-1	0	0	
GSM5976516	1	32	0	0	NaN	-1	0	0	
GSM5976596	1	46	0	0	NaN	0	2	0	
GSM5976607	1	54	0	1	NaN	0	3	2	
GSM5976608	1	54	6	0	NaN	0	3	2	
...	...	...	...	...	...	...	...	...	
GSM5978853	1	21	0	1	NaN	0	1	2	
GSM5978854	1	21	2	0	NaN	0	1	2	
GSM5978918	1	71	0	0	NaN	-1	-1	0	
GSM5978929	1	61	6	0	NaN	-1	-1	0	
GSM5978930	1	61	0	0	NaN	-1	-1	0	

89 rows × 43 columns

```
rows_with_na_noNA = X_noNA[X_noNA.isna().any(axis=1)].index


# Apply the function to replace NaNs
X_noNA = my_replace_na(X_noNA)
X_noNA.loc[rows_with_na_noNA]
```

ID	Sex	Age	Tissue.region	Inflamed.yn	log2_CRP	Clinically.Active	SESCD_Endoscopic.CD	Max.Nancy_Histo.UC	Max.GHAS_Histo.
GSM5976515	1	32	2	0	1.930563	-1	0	0	
GSM5976516	1	32	0	0	1.930563	-1	0	0	
GSM5976596	1	46	0	0	1.930563	0	2	0	
GSM5976607	1	54	0	1	1.930563	0	3	2	
GSM5976608	1	54	6	0	1.930563	0	3	2	
...	...	...	...	...	...	...	...	...	
GSM5978853	1	21	0	1	1.930563	0	1	2	
GSM5978854	1	21	2	0	1.930563	0	1	2	
GSM5978918	1	71	0	0	1.930563	-1	-1	0	
GSM5978929	1	61	6	0	1.930563	-1	-1	0	
GSM5978930	1	61	0	0	1.930563	-1	-1	0	

89 rows × 43 columns

```
# First SVM using cross validation:
### Note: kernels are the mathematical function that transforms data into a linear format so that
### SVMs can classify non-linear data, C controls the balance between maximizing the margin and
### minimizing misclassifications

svcl_05 = svm.SVC(kernel="linear", C=0.5, random_state=1)
svcl_1 = svm.SVC(kernel="linear", C=1, random_state=1)
svcl_15 = svm.SVC(kernel="linear", C=1.5, random_state=1)

svcr_05 = svm.SVC(kernel="rbf", C=0.5, random_state=1)
svcr_1 = svm.SVC(kernel="rbf", C=1, random_state=1)
svcr_15 = svm.SVC(kernel="rbf", C=1.5, random_state=1)


# Initialize the scaler
scaler = StandardScaler()
# Scale only the 'log2_CRP' column
X_rem2['log2_CRP'] = scaler.fit_transform(X_rem2[['log2_CRP']])
X_noNA['log2_CRP'] = scaler.fit_transform(X_noNA[['log2_CRP']])


# conduct cross validation: use the cross_val_score function to fit the specified SVC to my X and y,
# performing 5 iterations over the data with each svc by specifying cv=5
# extract an accuracy score for each time the specified SVC ran a training and validation (5 scores
# for the 5 iterations)

svcl_05_scores = cross_val_score(svcl_05, X_rem2, y_rem2, cv=5)
svcl_1_scores = cross_val_score(svcl_1, X_rem2, y_rem2, cv=5)
svcl_15_scores = cross_val_score(svcl_15, X_rem2, y_rem2, cv=5)

svcr_05_scores = cross_val_score(svcr_05, X_rem2, y_rem2, cv=5)
svcr_1_scores = cross_val_score(svcr_1, X_rem2, y_rem2, cv=5)
svcr_15_scores = cross_val_score(svcr_15, X_rem2, y_rem2, cv=5)


# noNA
svcl_05_scores_noNA = cross_val_score(svcl_05, X_noNA, y_noNA, cv=5)
svcl_1_scores_noNA = cross_val_score(svcl_1, X_noNA, y_noNA, cv=5)
svcl_15_scores_noNA = cross_val_score(svcl_15, X_noNA, y_noNA, cv=5)

svcr_05_scores_noNA = cross_val_score(svcr_05, X_noNA, y_noNA, cv=5)
svcr_1_scores_noNA = cross_val_score(svcr_1, X_noNA, y_noNA, cv=5)
svcr_15_scores_noNA = cross_val_score(svcr_15, X_noNA, y_noNA, cv=5)



# create a df for my results
# first create a dictionary assigning my colnames and info
# use the mean method to calculate my average score for each model so I can compare them more easily
myres = {
    'Dataframe' : ['rem2: 41 Features'] * 6 + ['noNA: 43 Features'] * 6,
    'SVM Combo' : ['svcl_05: Linear, C=0.5', 'svcl_1: Linear, C=1',
```

```
                   'svcl_15: Linear, C=1.5', 'svcr_05: Radial, C=0.5',
                   'svcr_1: Radial, C=1', 'svcr_15: Radial, C=1.5',

                   'svcl_05: Linear, C=0.5', 'svcl_1: Linear, C=1',
                   'svcl_15: Linear, C=1.5', 'svcr_05: Radial, C=0.5',
                   'svcr_1: Radial, C=1', 'svcr_15: Radial, C=1.5'],
    'Mean CV Score' : [svcl_05_scores.mean(), svcl_1_scores.mean(),
                       svcl_15_scores.mean(), svcr_05_scores.mean(),
                       svcr_1_scores.mean(), svcr_15_scores.mean(),

                       svcl_05_scores_noNA.mean(), svcl_1_scores_noNA.mean(),
                       svcl_15_scores_noNA.mean(), svcr_05_scores_noNA.mean(),
                       svcr_1_scores_noNA.mean(), svcr_15_scores_noNA.mean()]
}

myresdf = pd.DataFrame(myres) # place my dictionary into a df
myresdf
```

	Dataframe	SVM Combo	Mean CV Score
0	rem2: 41 Features	svcl_05: Linear, C=0.5	0.765289
1	rem2: 41 Features	svcl_1: Linear, C=1	0.762860
2	rem2: 41 Features	svcl_15: Linear, C=1.5	0.765693
3	rem2: 41 Features	svcr_05: Radial, C=0.5	0.633349
4	rem2: 41 Features	svcr_1: Radial, C=1	0.664103
5	rem2: 41 Features	svcr_15: Radial, C=1.5	0.687577
6	noNA: 43 Features	svcl_05: Linear, C=0.5	0.765690
7	noNA: 43 Features	svcl_1: Linear, C=1	0.765689
8	noNA: 43 Features	svcl_15: Linear, C=1.5	0.764878
9	noNA: 43 Features	svcr_05: Radial, C=0.5	0.630515
10	noNA: 43 Features	svcr_1: Radial, C=1	0.665725
11	noNA: 43 Features	svcr_15: Radial, C=1.5	0.684340

```
myresdf_l = loaded_objects['myresdf']
myresdf_l.to_excel('/241209_SVM.stats.xlsx', index=False)  # index=False avoids saving row indices


# does increasing C to 2 improve it? no
svcr_2 = svm.SVC(kernel="rbf", C=2, random_state=1)
svcr_2_scores = cross_val_score(svcr_2, X_rem2, y_rem2, cv=5)
svcr_2_scores.mean()
```

```
0.696080644501697
```

```
# best SVM model: 41 features, linear, C=1.5
## Note: removing the 2 features with 43% missing data only improved accuracy very slightly, neglibily

# now split into training and testing, then fit and predict with this best performer

# perform train test split, specifying test_size=0.2 so the test is 20%
# setting 4 new objects: X_train, X_test, y_train, y_test
X_rem2_train, X_rem2_test, y_rem2_train, y_rem2_test = train_test_split( X_rem2, y_rem2, test_size = 0.2)


len(X_rem2_train)
```

```
1976
```

```
len(X_rem2_test)
```

```
495
```

```
X_rem2_train.head()
```

ID	Sex	Age	Tissue.region	Inflamed.yn	log2_CRP	Clinically.Active	SESCD_Endoscopic.CD	X2346	X1116	X2215	...
GSM5977748	1	38	1	0	-0.728054	0	2	-0.676321	-0.310596	-0.778508	...
GSM5978103	0	56	3	0	0.858068	0	0	-0.130868	-0.797225	-0.551071	...
GSM5978196	0	41	2	1	0.772326	0	1	2.986257	-0.913558	-0.111811	...
GSM5976865	1	42	2	1	0.298178	1	1	3.930333	1.231347	3.226104	...
GSM5978577	1	64	0	0	-0.235538	-1	-1	-0.822266	-0.570684	-0.399475	...

5 rows × 41 columns

```
# using my most well performing model with a linear kernel and C=1.5, fit the SVC to my original
# X and y (because previously I conducted the cross_val_score, but had not actually fit it)
# use the fit method to train the specified SVM on the dataset
mysvm = svcl_15.fit(X_rem2_train, y_rem2_train)

# use the predict method to use the trained model to estimate y_hats for each patient after tmt
my_svm_pred = mysvm.predict(X_rem2_test)
my_svm_pred
```

```
array([2, 2, 1, 2, 1, 1, 1, 2, 0, 0, 2, 0, 0, 1, 0, 1, 2, 2, 0, 2, 1, 0,
       1, 1, 2, 1, 0, 2, 0, 2, 0, 2, 2, 0, 1, 0, 2, 2, 2, 0, 2, 1, 0, 0,
       2, 0, 0, 2, 0, 0, 2, 2, 2, 0, 1, 2, 0, 2, 2, 0, 0, 0, 2, 2, 0, 2,
       0, 1, 1, 2, 1, 0, 0, 1, 2, 2, 0, 1, 1, 2, 2, 0, 0, 1, 2, 0, 2, 0,
       1, 0, 0, 0, 1, 0, 2, 2, 2, 1, 0, 2, 1, 1, 0, 2, 2, 2, 0, 0, 0, 2,
       2, 1, 0, 0, 1, 0, 2, 0, 2, 2, 2, 0, 2, 1, 0, 2, 2, 0, 0, 2, 0, 0,
       2, 0, 2, 1, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 2, 2, 1, 0, 0, 2, 2, 2, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 2,
       0, 2, 0, 0, 1, 2, 0, 0, 1, 2, 2, 0, 2, 2, 2, 0, 1, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 2, 2, 0, 0, 0, 1, 2, 1,
       1, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0, 2, 1, 1, 1, 2, 2, 2, 2, 2, 0,
       0, 1, 2, 0, 2, 0, 2, 0, 0, 1, 2, 1, 0, 0, 2, 0, 2, 1, 1, 0, 2, 2,
       1, 2, 0, 0, 0, 2, 0, 1, 0, 0, 2, 0, 0, 0, 2, 0, 2, 0, 2, 1, 0, 0,
       0, 0, 2, 0, 0, 2, 2, 0, 0, 2, 0, 2, 1, 0, 1, 0, 0, 1, 2, 0, 0, 0,
       0, 2, 2, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 2, 2, 0, 0, 2, 2,
       0, 0, 2, 2, 2, 1, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 2, 2, 0, 2, 0, 0,
       0, 1, 0, 2, 0, 0, 2, 2, 0, 0, 0, 1, 2, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 1, 0, 2, 0, 0, 0, 1, 1, 2, 0, 2, 2, 0, 2, 2, 2,
       2, 1, 0, 1, 0, 2, 1, 0, 2, 1, 0, 2, 2, 0, 0, 0, 2, 1, 1, 0, 0, 1,
       0, 1, 2, 0, 0, 2, 2, 2, 2, 1, 2, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0,
       1, 0, 2, 0, 0, 2, 0, 2, 1, 0, 0, 2, 0, 2, 0, 0, 2, 0, 2, 2, 1, 2,
       0, 0, 0, 1, 0, 0, 2, 2, 0, 0, 0, 1, 1, 0, 2, 2, 0, 0, 2, 2, 0, 2,
       2, 0, 0, 0, 0, 0, 1, 2, 1, 2, 2])
```

```
accuracy_score(y_rem2_test, my_svm_pred)*100
```

```
79.1919191919192
```

```
X_noNA_train, X_noNA_test, y_noNA_train, y_noNA_test = train_test_split( X_noNA, y_noNA, test_size = 0.2)
mysvm_noNA = svcl_15.fit(X_noNA_train, y_noNA_train)
my_svm_pred_noNA = mysvm_noNA.predict(X_noNA_test)
accuracy_score(y_noNA_test, my_svm_pred_noNA)*100
```

```
75.95959595959596
```

```
## try removing the cols with any NA's:
X_rem5 = X_rem2.copy()
X_rem5 = X_rem5.drop(['log2_CRP', 'Clinically.Active', 'SESCD_Endoscopic.CD'], axis = 1)
X_rem5.head()
```

ID	Sex	Age	Tissue.region	Inflamed.yn	X2346	X1116	X2215	X29881	X6289	X6279	...	X4586	X4319
GSM5976499	0	44	0	0	-0.734807	-0.239503	-0.795064	-0.114424	-1.028815	-0.688270	...	0.816403	-0.609817
GSM5976500	0	44	1	0	-0.798520	-0.661091	-1.133618	-0.966807	-1.028815	-0.691740	...	-0.475171	-1.190500
GSM5976501	0	44	2	0	0.220419	0.206250	-0.501224	0.109697	-0.616697	-0.577078	...	-0.721310	-1.190500
GSM5976502	1	60	3	0	0.617282	-0.735794	-0.615965	-0.682069	-0.280800	-0.353916	...	-0.723736	-0.327821
GSM5976503	1	60	1	0	-0.618784	-0.662444	-0.820484	-0.441204	-0.307150	-0.619962	...	-0.684677	-0.160902

5 rows × 38 columns

```
# can use the same y_rem2 for y_rem5
y_rem5 = y_rem2.copy()
X_rem5_train, X_rem5_test, y_rem5_train, y_rem5_test = train_test_split( X_rem5, y_rem5, test_size = 0.2)
mysvm_rem5 = svcl_15.fit(X_rem5_train, y_rem5_train)
my_svm_pred5 = mysvm_rem5.predict(X_rem5_test)
accuracy_score(y_rem5_test, my_svm_pred5)*100
```

```
61.81818181818181
```

```
X_rem5_l = loaded_objects['X_rem5']
y_rem5_l = loaded_objects['y_rem5']
X_rem5_l.head()
```

ID	Sex	Age	Tissue.region	Inflamed.yn	X2346	X1116	X2215	X29881	X6289	X6279	...	X4586	X4319
GSM5976499	0	44	0	0	-0.734807	-0.239503	-0.795064	-0.114424	-1.028815	-0.688270	...	0.816403	-0.609817
GSM5976500	0	44	1	0	-0.798520	-0.661091	-1.133618	-0.966807	-1.028815	-0.691740	...	-0.475171	-1.190500
GSM5976501	0	44	2	0	0.220419	0.206250	-0.501224	0.109697	-0.616697	-0.577078	...	-0.721310	-1.190500
GSM5976502	1	60	3	0	0.617282	-0.735794	-0.615965	-0.682069	-0.280800	-0.353916	...	-0.723736	-0.327821
GSM5976503	1	60	1	0	-0.618784	-0.662444	-0.820484	-0.441204	-0.307150	-0.619962	...	-0.684677	-0.160902

5 rows × 38 columns

```
## SVM using 41 features (instead of replacing 43% NA's with median or mode), linear, C=1.5
## has 79% accuracy. Removing any columns with NA's reduces the accuracy by ~18% and keeping
## 2 cols with 43% imputed data reduced accuracy by 4%.
```

Decision Tree

```
# Now Decision Tree, can use noNA df since highest accuracy probability:

model = DecisionTreeClassifier(random_state=50)

# Set up the hyperparameter search space
param_dist = {
    'criterion':['gini','entropy'], # try both gini and entropy impurity measures
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_leaf': [1, 2, 5, 10],
    'min_samples_split': [2, 5, 10, 20],
    'max_features': ['sqrt', 'log2'] # diff ways to calculate the number of features
                                        # to consider when looking for the best split
}

# Set up RandomizedSearchCV instead of GridSearchCV: checks random combinations instead of
# every possible combination, determine the best combination of hyper parameters by
# evaluating model performance using cross-validation
# You can use StratifiedKFold for cross-validation to maintain class distribution
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=50)
# data is shuffled before splitting into folds to ensure the folds are more
# representative of the overall data

random_search = RandomizedSearchCV(
```

```
    estimator=model,
    param_distributions=param_dist,
    n_iter=50,
    cv=cv,
    verbose=1, # minimal output showing: hyperparameters being tested, cv results for
    # each combination, and how much time each combination is taking
    n_jobs=4,  # Uses 4 out of 11 available cores to speed up computation
    random_state=50
)



# Fit the RandomizedSearchCV to the training data
random_search.fit(X_rem2_train, y_rem2_train)


X_rem5_train_l = loaded_objects['X_rem5_train']
X_rem5_test_l = loaded_objects['X_rem5_test']
y_rem5_train_l = loaded_objects['y_rem5_train']
y_rem5_test_l = loaded_objects['y_rem5_test']


X_rem2_train_l = loaded_objects['X_rem2_train']
X_rem2_test_l = loaded_objects['X_rem2_test']
y_rem2_train_l = loaded_objects['y_rem2_train']
y_rem2_test_l = loaded_objects['y_rem2_test']


best_tree_l = loaded_objects['best_tree']


best_tree_l.score(X_rem2_train_l, y_rem2_train_l)
```

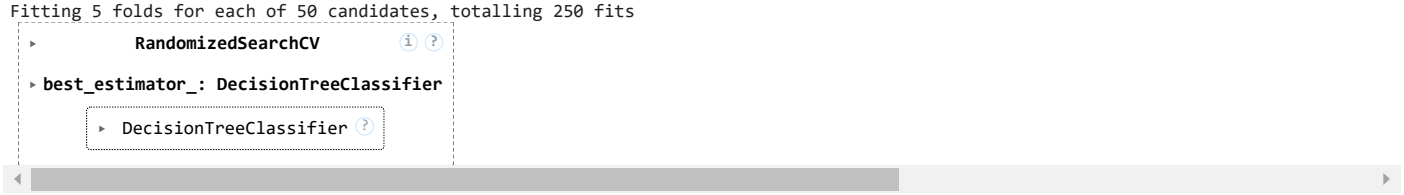
```
0.8901821862348178
```

```
best_tree_l.score(X_rem2_test_l, y_rem2_test_l)
```

```
0.7353535353535353
```

```
# Fit the RandomizedSearchCV to the training data
random_search.fit(X_rem5_train_l, y_rem5_train_l)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
```




```
# for_rem5
print(random_search.best_score_) # highest accuracy
print(random_search.best_params_) # best combination of hyper parameters
print(random_search.best_estimator_) # this is the best model
```

```
0.5718693261731237
{'min_samples_split': 20, 'min_samples_leaf': 2, 'max_features': 'log2', 'max_depth': 10, 'criterion': 'entropy'}
DecisionTreeClassifier(criterion='entropy', max_depth=10, max_features='log2',
                       min_samples_leaf=2, min_samples_split=20,
                       random_state=50)
```

```
best_tree_rem5 = random_search.best_estimator_


# for rem_2
print(random_search.best_score_) # highest accuracy
print(random_search.best_params_) # best combination of hyper parameters
print(random_search.best_estimator_) # this is the best model
```

```
0.7145633550696842
{'min_samples_split': 20, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': None, 'criterion': 'entropy'}
DecisionTreeClassifier(criterion='entropy', max_features='sqrt',
                       min_samples_split=20, random_state=50)
```

```
# set the best estimator to an object and train this object
best_tree = random_search.best_estimator_
```

```
# best_tree_fitted = best_tree.fit(X_rem2_train, y_rem2_train) # redundant, best_tree can be used to predict
best_tree
```

```
DecisionTreeClassifier(criterion='entropy', max_features='sqrt',
                       min_samples_split=20, random_state=50)
```

```
# use the predict method to use my model on the test data
best_tree_pred = best_tree.predict(X_rem2_test)
accuracy_score(y_rem2_test,best_tree_pred)*100
```

```
73.53535353535354
```

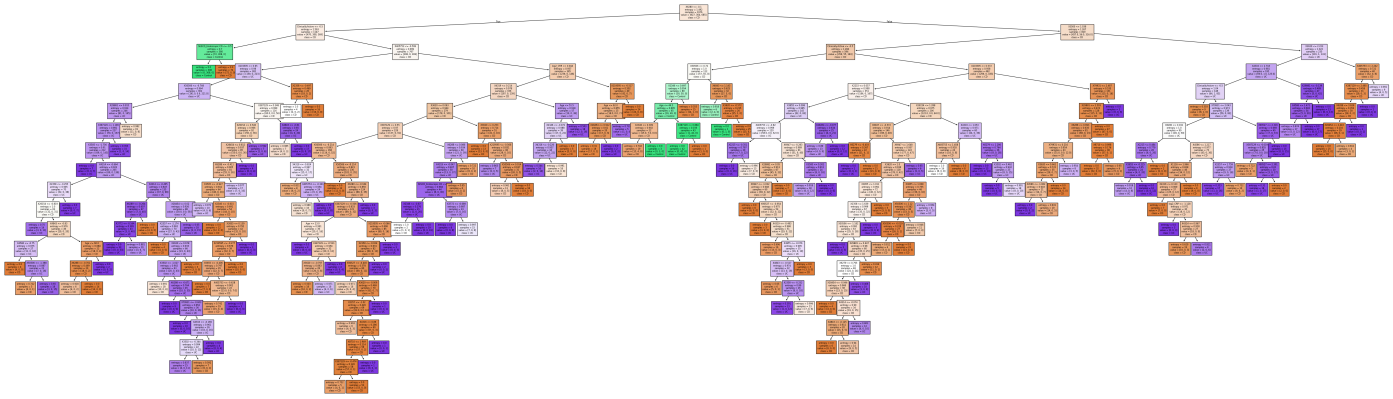
```
best_tree_fitted_l = loaded_objects['best_tree_fitted']
best_tree_fitted_l.get_depth()
```

```
19
```

```
type(best_tree_fitted_l)
```

```
sklearn.tree._classes.DecisionTreeClassifier
def __init__(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None,
max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0,
monotonic_cst=None)

A decision tree classifier.

Read more in the :ref:`User Guide <tree>`.

Parameters
----------
```

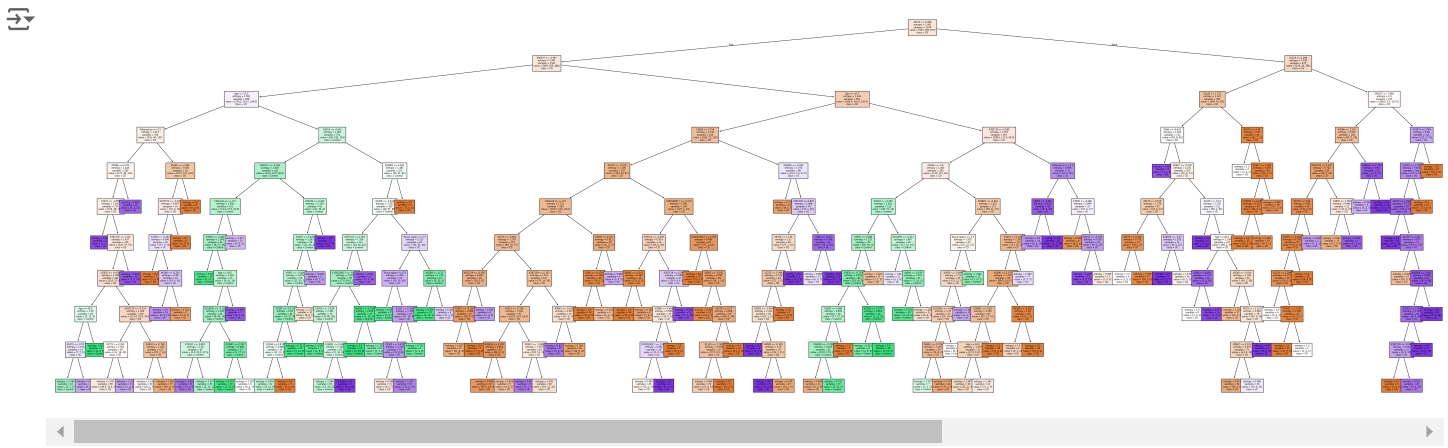
```
# use the figure method to set the dimensions of my plot
plt.figure(figsize=(70, 20))

# Plot the tree and color the nodes based on class
plot_tree(best_tree_fitted,
          feature_names=X_rem2.columns,  # List of feature names
          class_names=vals_rem2,  # List of class names
          filled=True,  # Color nodes based on class
          fontsize=6)  # Adjust font size for better readability

# Save the tree plot to a PDF
plt.savefig("241208_best.dec.tree.pdf", format="pdf", bbox_inches="tight")

# Show the plot if needed
plt.show()
```



```
X_rem2_l = loaded_objects['X_rem2']
vals_rem2_l = loaded_objects['vals_rem2']
```

```
vals_rem5_l = loaded_objects['vals_rem2']
```

```
# use the figure method to set the dimensions of my plot
plt.figure(figsize=(70, 20))

# Plot the tree and color the nodes based on class
plot_tree(best_tree_rem5,
          feature_names=X_rem5_l.columns,  # List of feature names
          class_names=vals_rem5_l,  # List of class names
          filled=True,  # Color nodes based on class
          fontsize=6)  # Adjust font size for better readability

# Save the tree plot to a PDF
plt.savefig("241209_best.dec.tree.pdf", format="pdf", bbox_inches="tight")

# Show the plot if needed
plt.show()
```



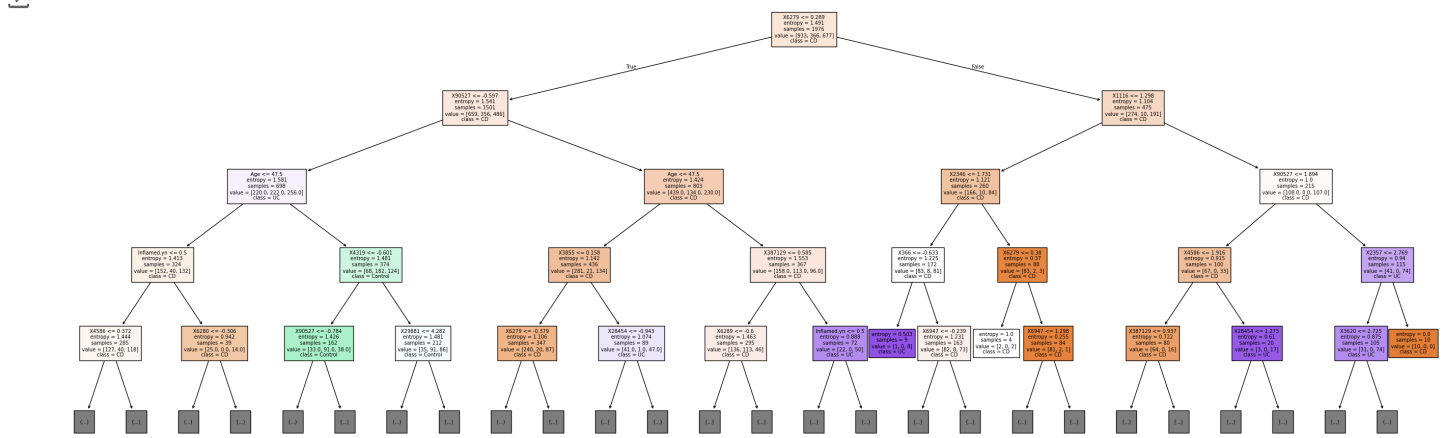
```
best_tree_rem5.get_depth()
```

```
10
```

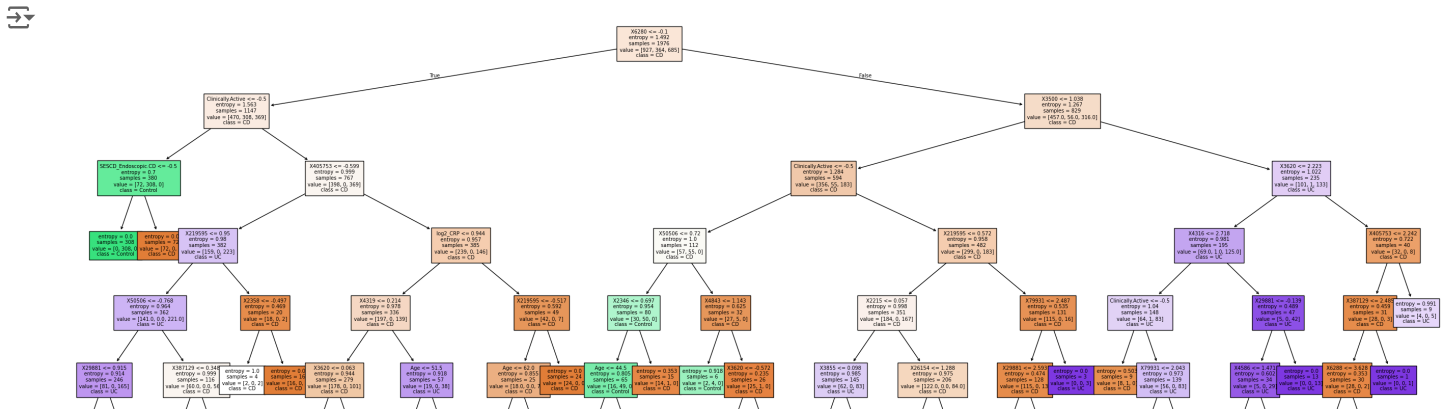
```
vals_rem5_l
```

```
array(['CD', 'Control', 'UC'], dtype=object)
```

```
plt.figure(figsize=(35, 12))  # width, height
plot_tree(best_tree_rem5, max_depth=4,
          filled=True, feature_names=X_rem5_l.columns,
          class_names=vals_rem5_l,
          fontsize=7)

# Save the tree plot to a PDF
plt.savefig("241209_actual.dec.tree.cut4.pdf", format="pdf", bbox_inches="tight")

plt.show()
```



```
plt.figure(figsize=(35, 12))  # width, height
plot_tree(best_tree_fitted_1, max_depth=5,
          filled=True, feature_names=X_rem2_1.columns,
          class_names=vals_rem2_1,
          fontsize=7)

# Save the tree plot to a PDF
plt.savefig("241209_dec.tree.cut5.pdf", format="pdf", bbox_inches="tight")

plt.show()
```



```
param_dist = {
    'criterion':['gini','entropy'], # try both gini and entropy impurity measures
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_leaf': [1, 2, 5, 10],
    'min_samples_split': [2, 5, 10, 20],
    'max_features': ['sqrt', 'log2'] # diff ways to calculate the number of features
                                     # to consider when looking for the best split
}

# Set up RandomizedSearchCV instead of GridSearchCV: checks random combinations instead of
# every possible combination, determine the best combination of hyper parameters by
# evaluating model performance using cross-validation
# StratifiedKFold for cross-validation: data is shuffled before splitting into folds
# to ensure the folds are more representative of the overall data
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=50)

random_search_rf = RandomizedSearchCV(
    estimator=model_rf,
    param_distributions=param_dist,
    n_iter=50,
    cv=cv,
    verbose=1, # minimal output showing: hyperparameters being tested, cv results for
    # each combination, and how much time each combination is taking
    n_jobs=4,  # Uses 4 out of 11 available cores to speed up computation
    random_state=50
)
```

Random Forest

```
# random forest
model_rf = RandomForestClassifier(oob_score = True, random_state=50)

# Set up the hyperparameter search space
param_dist = {
    'criterion':['gini','entropy'], # try both gini and entropy impurity measures
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_leaf': [1, 2, 5, 10],
    'min_samples_split': [2, 5, 10, 20],
    'max_features': ['sqrt', 'log2'] # diff ways to calculate the number of features
                                     # to consider when looking for the best split
}

# Set up RandomizedSearchCV instead of GridSearchCV: checks random combinations instead of
# every possible combination, determine the best combination of hyper parameters by
# evaluating model performance using cross-validation
# You can use StratifiedKFold for cross-validation to maintain class distribution
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=50)
```

```
# data is shuffled before splitting into folds to ensure the folds are more
# representative of the overall data

random_search_rf = RandomizedSearchCV(
    estimator=model_rf,
    param_distributions=param_dist,
    n_iter=50,
    cv=cv,
    verbose=1, # minimal output showing: hyperparameters being tested, cv results for
    # each combination, and how much time each combination is taking
    n_jobs=4,  # Uses 4 out of 11 available cores to speed up computation
    random_state=50
)

# Fit the RandomizedSearchCV to the training data
random_search_rf.fit(X_rem2_train, y_rem2_train)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
```

RandomizedSearchCV

best_estimator_: RandomForestClassifier

RandomForestClassifier

```
print(random_search_rf.best_score_) # highest accuracy
print(random_search_rf.best_params_) # best combination of hyper parameters
print(random_search_rf.best_estimator_) # this is the best model
```

```
0.7894540340109961
{'min_samples_split': 2, 'min_samples_leaf': 5, 'max_features': 'log2', 'max_depth': 20, 'criterion': 'gini'}
RandomForestClassifier(max_depth=20, max_features='log2', min_samples_leaf=5,
                       oob_score=True, random_state=50)
```

```
# set the best estimator to an object and train this object
best_rf = random_search_rf.best_estimator_
best_rf_fitted = best_rf.fit(X_rem2_train, y_rem2_train)
best_rf_fitted
```

RandomForestClassifier

```
RandomForestClassifier(max_depth=20, max_features='log2', min_samples_leaf=5,
                       oob_score=True, random_state=50)
```

```
# the oob score returns the accuracy of the classifier based on the out-of-bag samples
# oob estimates of how well the model generalizes
# out-of-bag samples are data points from the original training set that were not selected for
# a given tree due to the bootstrap nature of random forest
# testing the predictions from these samples acts as a makeshift internal cross validation
best_rf.oob_score_
```

```
0.7813765182186235
```

```
# checking the prediction accuracy on the test set with predict and accuracy_score
y_pred_rf2 = best_rf_fitted.predict(X_rem2_test)
accuracy_score(y_rem2_test, y_pred_rf2)
```

```
0.8141414141414142
```

```
# checking the prediction accuracy on the test set with predict and accuracy_score
y_pred_rf = best_rf.predict(X_rem2_test)
accuracy_score(y_rem2_test, y_pred_rf)
```

```
0.8141414141414142
```

AdaBoost

```
# Create adaboost classifer object and try different parameters to see its effect
abc = AdaBoostClassifier(n_estimators=100, # max number of estimators at which boosting is terminated
                         learning_rate=1, # Weight applied to each classifier at each boosting iteration
                         random_state=50,
                         estimator=best_rf,
```

```
                         algorithm='SAMME')
# Train Adaboost Classifer with fit method
model_rf_boosted = abc.fit(X_rem2_train, y_rem2_train)


# Predict the response for test dataset and check accuracy
y_pred_rf_boosted = model_rf_boosted.predict(X_rem2_test)
accuracy_score(y_rem2_test, y_pred_rf_boosted)
```

```
0.8121212121212121
```

```
# Create adaboost classifer object and try different parameters to see its effect
abc2 = AdaBoostClassifier(n_estimators=100, # max number of estimators at which boosting is terminated
                         learning_rate=1.5, # Weight applied to each classifier at each boosting iteration
                         random_state=50,
                         estimator=best_rf,
                         algorithm='SAMME')
# Train Adaboost Classifer with fit method
model_rf_boosted2 = abc2.fit(X_rem2_train, y_rem2_train)

# Predict the response for test dataset and check accuracy
y_pred_rf_boosted2 = model_rf_boosted2.predict(X_rem2_test)
accuracy_score(y_rem2_test, y_pred_rf_boosted2)
```

```
0.8202020202020202
```

```
# Create adaboost classifer object and try different parameters to see its effect
abc3 = AdaBoostClassifier(n_estimators=150, # max number of estimators at which boosting is terminated
                         learning_rate=1, # Weight applied to each classifier at each boosting iteration
                         random_state=50,
                         estimator=best_rf,
                         algorithm='SAMME')
# Train Adaboost Classifer with fit method
model_rf_boosted3 = abc3.fit(X_rem2_train, y_rem2_train)

# Predict the response for test dataset and check accuracy
y_pred_rf_boosted3 = model_rf_boosted3.predict(X_rem2_test)
accuracy_score(y_rem2_test, y_pred_rf_boosted3)
```

```
0.802020202020202
```

```
# Create adaboost classifer object and try different parameters to see its effect
abc4 = AdaBoostClassifier(n_estimators=100, # max number of estimators at which boosting is terminated
                         learning_rate=2, # Weight applied to each classifier at each boosting iteration
                         random_state=50,
                         estimator=best_rf,
                         algorithm='SAMME')
# Train Adaboost Classifer with fit method
model_rf_boosted4 = abc4.fit(X_rem2_train, y_rem2_train)

# Predict the response for test dataset and check accuracy
y_pred_rf_boosted4 = model_rf_boosted4.predict(X_rem2_test)
accuracy_score(y_rem2_test, y_pred_rf_boosted4)
```

```
0.8080808080808081
```

```
## The first AdaBoost was the best, yielding 82% prediction accuracy:
### on 41 features (removing instead of replacing 43% NA's with median or mode), 3 of
### the features had 3-26% data replaced with median or mode
### RandomForestClassifier(max_depth=20, max_features='log2', min_samples_leaf=5,
###                        oob_score=True, random_state=50)
### abc2 = AdaBoostClassifier(n_estimators=100, # max number of estimators at which boosting is terminated
###                           learning_rate=1.5, # Weight applied to each classifier at each boosting iteration
###                           random_state=50,
###                           estimator=best_rf,
###                           algorithm='SAMME')


type(my_svm_pred)
```

```
numpy.ndarray
```

XGBoost

```
# R used for XGBoost sections
# Format dataframe for use in XGBoost
# perform VST and normalize data
library(DESeq2)
vst_data = varianceStabilizingTransformation(dds, blind = TRUE)
vst_matrix = assay(vst_data)
z_score_normalized = apply(vst_matrix, 1, function(x) (x - mean(x)) / sd(x))
# put normalized counts back in df with metadata
top_expr = as.data.frame(z_score_normalized[,top_genes])
model_data = cbind(final.meta, top_expr)


# Select genes for use in model
# Get the number of observations for each class
num_UC = sum(model_data$Disease.type == "UC")
num_CD = sum(model_data$Disease.type == "CD")
num_Control = sum(model_data$Disease.type == "Control")
# Define the number of gene features
num_feat = 35

# Get the number of genes from each disease

# Define the exact nunmber of each genes from UC and CD
mostsig_UC = round(num_feat / 2)
mostsig_CD = num_feat - mostsig_UC

# Get that number of genes
UC_genes = row.names(pval_UC_sort)[1:mostsig_UC]
CD_genes = row.names(pval_CD_sort)[1:mostsig_CD]
unique_genes = unique(UC_genes, CD_genes)

# Get a list of genes equal to the number specified, accounting for genes
# that appear in both diseases
while (length(unique_genes) < num_feat){
  mostsig_UC = mostsig_UC+1
  mostsig_CD = mostsig_CD+1
  next_UC_gene = row.names(pval_UC_sort)[mostsig_UC]
  next_CD_gene = row.names(pval_CD_sort)[mostsig_CD]
  unique_genes = unique(c(unique_genes,next_CD_gene, next_UC_gene))
  if (length(unique_genes) > num_feat){
    unique_genes = unique_genes[-1]
  }
}

# Add genes and selected metadata to dataframe
features = c(colnames(model_data)[1:5],unique_genes)
feat_data = model_data[,paste0("X",features[6:length(features)])]
feat_data = cbind(model_data[,features[1:5]],feat_data)
labels = model_data$Disease.type
feat_data$Disease.type = as.factor(feat_data$Disease.type)
feat_data$Sex = as.factor(feat_data$Sex)
feat_data$Tissue.region = as.factor(feat_data$Tissue.region)
feat_data$Inflamed.yn = as.factor(feat_data$Inflamed.yn)


library(xgboost)
library(caret)

#Make labels numeric (Eg. 0, 1 ,2)
labels = as.numeric(factor(feat_data$Disease.type, levels = c("Control","CD","UC")))-1

# Do One-hot encoding for categorical variables
encoded_sex = model.matrix(~ Sex - 1, data = feat_data)
encoded_tis = model.matrix(~ Tissue.region - 1, data = feat_data)
encoded_inf = model.matrix(~ Inflamed.yn - 1, data = feat_data)
encoded_df = cbind(encoded_sex,encoded_tis,encoded_inf,feat_data[,-c(1:5)])

# Make training test splits
train_id = createDataPartition(feat_data$Disease.type, p = 0.8, list = FALSE)
train_data = encoded_df[train_id,-1]
test_data = encoded_df[-train_id,-1]
train_label = labels[train_id]
test_label = labels[-train_id]
data_matrix = as.matrix(train_data)

# Make XGB matrix object for training
train_xgb_matrix = xgb.DMatrix(
  data = as.matrix(train_data),
```

```
  label = train_label,
  missing = NA)

# And testing
test_xgb_matrix = xgb.DMatrix(
  data = as.matrix(test_data),
  label = test_label,
  missing = NA)


# Optimize model parameters
library(caret)
# Make a grid of parameters to test
params_grid = expand.grid(
  nrounds = c(2, 5, 10),
  eta = c(0.01, 0.3, 0.5),
  max_depth = c(3, 6, 9),
  gamma = c(0, 1, 5),
  min_child_weight = c(1, 5, 10),
  subsample = c(0.5, 0.75, 1),
  colsample_bytree = c(0.5, 0.75, 1)
)
# Define train control parameter
train_control = trainControl(
  method = 'cv',
  number = 5,
  verboseIter = F,
  allowParallel = T
)
options(warn = -1)
# Use grid search
  xgb_grid = suppressWarnings(train(
  x = as.matrix(train_df[1:500,]),
  y = as.factor(train_label[1:500]),
  method = "xgbTree",
  trControl = train_control,
  tuneGrid = params_grid,
  metric = "Accuracy",
  verbose = 0
  ))


xgb_grid$bestTune


# Evaluate on test set
params = list(
  max_depth =10,
  eta = 0.3,
  gamma = 1,
  colsample_bytree = 1,
  min_child_weight = 5,
  subsample = 1)
nrounds = 15
xgb_model = xgboost(
  data = train_xgb_matrix,
  params = params,
  nrounds = nrounds,
  verbose = 1,
  objective = "multi:softmax",
  num_class = 3)

# Test
pred = predict(xgb_model, test_xgb_matrix)
accuracy = sum(pred == test_label) / length(test_label)
print(paste("multi-class Accuracy:", round(accuracy * 100, 2), "%"))



# Generate ROC curves
library(pROC)
library(xgboost)
# Initialize variables
roc_curves = list()
auc_values = numeric()

# Train model again, this time getting probabilities
xgb_model = xgboost(
```

```
  data = train_xgb_matrix,
  params = params,
  nrounds = nrounds,
  verbose = 1,
  num_class = 3,
  objective = "multi:softprob")
# Predict probabilities on the test set
pred_probs = predict(xgb_model, test_xgb_matrix)
pred_probs_matrix = matrix(pred_probs, ncol = 3, byrow = TRUE)

for (class in c(0, 1, 2)) {
  # Create binary labels for one-vs-all
  binary_labels = ifelse(test_label == class, 1, 0)

  # Get probabilities for the current class
  class_probs = pred_probs_matrix[, class + 1]

  # Get ROC and AUC
  roc_curves[[class + 1]] = roc(binary_labels, class_probs)
  auc_values[class + 1] = auc(roc_curves[[class + 1]])
}

# Plot ROC curves for all classes
plot(roc_curves[[1]], col = "red", main = "ROC Curves using XGBoost")
plot(roc_curves[[2]], col = "blue", add = TRUE)
plot(roc_curves[[3]], col = "green", add = TRUE)

# Add a legend
legend("bottomright", legend = c("Control", "CD", "UC"), col = c("red", "blue", "green"), lwd = 2)


# Get metrics
library(caret)
conf_matrix = confusionMatrix(as.factor(pred), as.factor(test_label))
precision = conf_matrix$byClass[, "Precision"]
recall = conf_matrix$byClass[, "Recall"]
f1 = conf_matrix$byClass[, "F1"]
metrics = data.frame(
  Precision = precision,
  Recall = recall,
  F1 = f1,
  AUC = auc_values
)
row.names(metrics) = c("Control","CD","UC")
library(knitr)
kable(metrics, caption = "Evaluation Metrics", digits = 2)
```

Show hidden output

Next steps: Explain error

Pickle and Metrics

```
import pickle

# Specify the objects you want to save
workspace = {name: value for name, value in globals().items()
             if isinstance(value, (pd.DataFrame, np.ndarray, DecisionTreeClassifier,
                                   RandomForestClassifier, AdaBoostClassifier, SVC, LinearSVC))}

# Save the selected objects to a pickle file
with open('/content/241208_workspace.pkl', 'wb') as f:
    pickle.dump(workspace, f)

## This is unnecessary, just downloading it from the file list is much faster
# Download the pickle file (for Google Colab)
# from google.colab import files
# files.download('/content/241208_workspace.pkl')
```

```
# re-access the workspace:
import pickle
with open('/Users/collad04/Downloads/241208_workspace.pkl', 'rb') as f:
    loaded_workspace = pickle.load(f)
```

```
## example of re-loading
y_rem2_test = loaded_workspace['y_rem2_test']
y_pred_rf_boosted2 = loaded_workspace['y_pred_rf_boosted2']

accuracy_score(y_rem2_test, y_pred_rf_boosted2)
```

Show hidden output

```
## To re-access the workspace:

# Re-upload the pickle file
from google.colab import files
uploaded = files.upload()

# Load the objects from the pickle file
with open('241208_workspace.pkl', 'rb') as f:
    loaded_objects = pickle.load(f)

## Accessing the objects from the loaded workspace
# random_search_rf_loaded = loaded_workspace['random_search_rf']
# X_rem2_loaded = loaded_workspace['X_rem2']
# y_rem2_loaded = loaded_workspace['y_rem2']
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable

```
y_rem2_test_l = loaded_objects['y_rem2_test']
   # myresdf, # dataframe of SVM trials
    # mysvm,
my_svm_pred_l = loaded_objects['my_svm_pred'] # fitted svm and prediction result
    # best_tree_fitted,
best_tree_pred_l = loaded_objects['best_tree_pred'] # fitted dec tree and prediction
    # best_rf_fitted,
y_pred_rf2_l = loaded_objects['y_pred_rf2'] # fitted random forest and prediction
    # model_rf_boosted2,
y_pred_rf_boosted_l = loaded_objects['y_pred_rf_boosted2']


y_rem2_test_l
```

```
array([2, 0, 1, 2, 1, 1, 1, 0, 2, 0, 2, 0, 2, 1, 0, 1, 0, 0, 0, 0, 1, 0,
       1, 1, 2, 1, 0, 2, 0, 2, 0, 2, 2, 0, 1, 2, 2, 2, 2, 0, 2, 1, 2, 0,
       2, 2, 0, 2, 0, 0, 2, 0, 2, 2, 1, 2, 0, 2, 2, 0, 0, 0, 2, 2, 0, 2,
       0, 1, 1, 2, 1, 0, 0, 1, 0, 2, 0, 1, 1, 2, 2, 0, 0, 1, 2, 0, 2, 0,
       1, 0, 0, 2, 1, 0, 2, 2, 2, 1, 2, 2, 1, 1, 0, 0, 2, 2, 0, 0, 0, 2,
       2, 1, 0, 2, 1, 2, 2, 0, 2, 2, 2, 2, 1, 0, 2, 0, 0, 0, 2, 0, 0,
       0, 0, 0, 1, 0, 1, 2, 2, 0, 0, 0, 2, 2, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 2, 1, 2, 0, 2, 2, 2, 1, 2, 0, 0, 1, 0, 0, 1, 1, 2, 0, 2,
       0, 2, 0, 2, 1, 2, 0, 0, 1, 2, 0, 0, 2, 2, 2, 0, 1, 0, 2, 0, 0, 0,
       0, 0, 2, 0, 2, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 2, 0, 0, 2, 1, 0, 1,
       1, 0, 2, 2, 0, 0, 0, 0, 2, 2, 0, 0, 0, 2, 1, 1, 1, 2, 2, 2, 2, 0,
       2, 1, 0, 0, 2, 0, 0, 0, 2, 1, 2, 1, 0, 0, 2, 0, 2, 1, 1, 0, 2, 2,
       1, 0, 0, 2, 2, 0, 0, 1, 0, 2, 2, 2, 0, 0, 0, 0, 2, 0, 2, 1, 0, 2,
       0, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 2, 0, 0, 0,
       0, 0, 2, 0, 2, 0, 1, 1, 0, 0, 0, 0, 2, 1, 1, 0, 2, 2, 0, 0, 2, 2,
       0, 2, 2, 2, 2, 1, 0, 0, 2, 1, 2, 0, 0, 0, 0, 2, 2, 2, 0, 2, 0, 0,
       2, 1, 2, 2, 0, 0, 2, 0, 0, 0, 0, 1, 2, 2, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 2, 0, 1, 2, 0, 1, 2, 2, 0, 2, 0, 1, 1, 0, 2, 2, 0, 0, 2, 2, 0,
       2, 1, 0, 1, 2, 0, 1, 0, 0, 1, 2, 0, 0, 2, 0, 0, 2, 1, 1, 2, 0, 1,
       0, 1, 0, 0, 0, 0, 2, 2, 0, 1, 2, 0, 2, 2, 0, 2, 2, 2, 2, 2, 0, 0,
       1, 0, 2, 2, 2, 2, 2, 2, 1, 0, 2, 2, 0, 2, 0, 2, 0, 2, 2, 2, 1, 0,
       2, 0, 0, 1, 0, 0, 2, 2, 0, 2, 0, 1, 1, 0, 2, 0, 0, 2, 0, 2, 0, 2,
       2, 0, 0, 0, 0, 2, 1, 0, 1, 2, 2])
```

```
my_svm_pred_l
```

```
array([2, 2, 1, 2, 1, 1, 1, 2, 0, 0, 2, 0, 0, 1, 0, 1, 2, 2, 0, 2, 1, 0,
       1, 1, 2, 1, 0, 2, 0, 2, 0, 2, 2, 0, 1, 0, 2, 2, 2, 0, 2, 1, 0, 0,
       2, 0, 0, 2, 0, 0, 2, 2, 2, 0, 1, 2, 0, 2, 2, 0, 0, 0, 2, 2, 0, 2,
       0, 1, 1, 2, 1, 0, 0, 1, 2, 2, 0, 1, 1, 2, 2, 0, 0, 1, 2, 0, 2, 0,
       1, 0, 0, 0, 1, 0, 2, 2, 2, 1, 0, 2, 1, 1, 0, 2, 2, 2, 0, 0, 0, 2,
       2, 1, 0, 0, 1, 0, 2, 0, 2, 2, 2, 0, 2, 1, 0, 2, 2, 0, 0, 2, 0, 0,
       2, 0, 2, 1, 0, 1, 2, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 2, 2, 1, 0, 0, 2, 2, 2, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 2,
       0, 2, 0, 0, 1, 2, 0, 0, 1, 2, 2, 0, 2, 2, 2, 0, 1, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 2, 2, 0, 0, 0, 1, 2, 1,
       1, 0, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 1, 1, 1, 2, 2, 2, 2, 0,
       0, 1, 2, 0, 2, 0, 2, 0, 0, 1, 2, 1, 0, 0, 2, 0, 2, 1, 1, 0, 2, 2,
       1, 2, 0, 0, 0, 2, 0, 1, 0, 0, 2, 0, 0, 0, 2, 0, 2, 0, 2, 1, 0, 0,
```

```
0, 0, 2, 0, 0, 2, 2, 0, 0, 2, 0, 2, 1, 0, 1, 0, 0, 1, 2, 0, 0, 0,
0, 2, 2, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 2, 2, 0, 0, 2, 2,
0, 0, 2, 2, 2, 1, 0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 2, 2, 0, 2, 0, 0,
0, 1, 0, 2, 0, 0, 2, 2, 0, 0, 0, 1, 2, 0, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 1, 0, 0, 1, 0, 2, 0, 0, 0, 1, 1, 2, 0, 2, 2, 0, 2, 2, 2,
2, 1, 0, 1, 0, 2, 1, 0, 2, 1, 0, 2, 2, 0, 0, 0, 2, 1, 1, 0, 0, 1,
0, 1, 2, 0, 0, 2, 2, 2, 2, 1, 2, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0,
1, 0, 2, 0, 0, 2, 0, 2, 1, 0, 0, 2, 0, 2, 0, 0, 2, 0, 2, 2, 1, 2,
0, 0, 0, 1, 0, 0, 2, 2, 0, 0, 0, 1, 1, 0, 2, 2, 0, 0, 2, 2, 0, 2,
2, 0, 0, 0, 0, 0, 1, 2, 1, 2, 2])
```

best_tree_pred_1

```
array([2, 0, 1, 2, 1, 1, 1, 0, 2, 0, 2, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 2, 0, 2, 2, 0, 1, 0, 0, 2, 2, 0, 2, 1, 2, 2,
       2, 0, 0, 2, 0, 0, 2, 2, 2, 2, 1, 2, 0, 0, 2, 0, 0, 0, 0, 2, 0, 2,
       0, 1, 1, 2, 1, 2, 0, 1, 0, 2, 0, 1, 1, 2, 0, 0, 0, 1, 2, 2, 2, 2,
       1, 0, 0, 0, 1, 0, 2, 0, 2, 1, 2, 2, 1, 1, 0, 2, 0, 2, 0, 0, 0, 2,
       0, 1, 0, 2, 1, 0, 0, 0, 0, 2, 2, 2, 2, 1, 2, 0, 0, 2, 0, 2, 0, 0,
       0, 0, 2, 0, 0, 1, 0, 2, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 2, 2, 1, 2, 0, 2, 0, 2, 1, 2, 0, 0, 1, 0, 2, 1, 1, 2, 2, 0,
       0, 0, 2, 0, 1, 0, 0, 0, 1, 2, 2, 0, 2, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 2, 2, 0, 2, 0, 0, 1, 1, 0, 2, 2, 2, 0, 1, 0, 1,
       1, 0, 2, 2, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 1, 1, 1, 0, 2, 2, 0, 0,
       2, 1, 2, 2, 2, 0, 0, 1, 0, 1, 0, 1, 0, 2, 2, 0, 2, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 2, 0, 0, 0, 2, 2, 0, 0, 2, 2, 0, 2, 2, 2, 1, 2, 0,
       0, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0, 2, 1, 0, 1, 0, 0, 1, 2, 0, 0, 2,
       0, 0, 0, 2, 2, 0, 1, 1, 2, 0, 0, 0, 2, 1, 1, 0, 0, 0, 2, 0, 2, 0,
       0, 2, 2, 2, 2, 1, 0, 2, 0, 1, 2, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 0,
       2, 1, 2, 2, 0, 0, 2, 0, 2, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 2, 0, 1, 0, 0, 1, 2, 2, 0, 0, 0, 1, 1, 2, 2, 0, 2, 2, 2, 0, 2,
       2, 1, 0, 1, 2, 2, 1, 0, 2, 1, 0, 2, 0, 0, 0, 2, 0, 1, 1, 2, 0, 1,
       0, 1, 2, 0, 2, 0, 0, 0, 2, 1, 2, 0, 2, 0, 0, 2, 0, 2, 2, 2, 0, 0,
       1, 0, 0, 2, 2, 0, 2, 2, 1, 2, 2, 2, 0, 0, 0, 2, 0, 2, 0, 2, 1, 2,
       2, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 1, 1, 0, 0, 2, 0, 0, 2, 2, 2, 2,
       2, 0, 0, 1, 0, 2, 1, 2, 1, 2, 0])
```

y_pred_rf2_1

```
array([0, 2, 1, 2, 1, 1, 1, 0, 2, 0, 2, 0, 2, 1, 2, 1, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 2, 2, 0, 1, 0, 0, 2, 2, 0, 0, 1, 0, 2,
       2, 0, 0, 2, 0, 0, 2, 2, 2, 0, 1, 2, 2, 0, 2, 0, 0, 0, 2, 2, 0, 2,
       0, 1, 1, 2, 1, 0, 0, 1, 2, 2, 0, 1, 1, 2, 0, 0, 0, 1, 2, 2, 2, 0,
       1, 0, 0, 0, 1, 0, 2, 0, 2, 1, 2, 0, 1, 1, 0, 2, 0, 2, 0, 0, 0, 2,
       2, 1, 0, 0, 1, 2, 2, 0, 2, 2, 2, 0, 2, 1, 0, 2, 0, 0, 0, 2, 0, 0,
       2, 0, 2, 1, 0, 1, 2, 2, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 2, 2, 1, 2, 0, 2, 2, 2, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 2,
       0, 2, 2, 2, 1, 2, 2, 0, 1, 2, 0, 0, 2, 2, 0, 0, 1, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 1, 1, 0, 2, 0, 0, 2, 1, 2, 1,
       1, 0, 2, 2, 0, 0, 2, 0, 2, 0, 0, 0, 0, 2, 1, 1, 1, 2, 2, 2, 0, 0,
       2, 1, 2, 0, 2, 0, 0, 0, 0, 1, 2, 1, 0, 0, 2, 0, 2, 1, 1, 0, 0, 2,
       1, 0, 0, 0, 0, 2, 0, 1, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 2, 1, 0, 2,
       0, 0, 2, 0, 0, 2, 2, 0, 0, 2, 2, 2, 1, 0, 1, 0, 0, 1, 2, 2, 0, 0,
       0, 0, 2, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 2, 1, 1, 0, 2, 0, 0, 2, 2, 2,
       0, 0, 2, 2, 2, 1, 0, 0, 2, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0,
       2, 1, 2, 2, 0, 0, 2, 0, 0, 0, 0, 1, 2, 2, 2, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 1, 0, 2, 0, 0, 0, 1, 1, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 1, 0, 1, 2, 2, 1, 0, 2, 1, 2, 0, 0, 0, 0, 0, 2, 1, 1, 0, 0, 1,
       0, 1, 2, 2, 0, 0, 2, 0, 0, 1, 2, 2, 2, 2, 2, 2, 0, 0, 2, 0, 0, 0,
       1, 0, 0, 2, 0, 0, 0, 2, 1, 0, 2, 2, 0, 2, 0, 2, 2, 0, 0, 2, 1, 0,
       0, 0, 0, 1, 0, 0, 2, 2, 0, 0, 0, 1, 1, 0, 2, 2, 0, 2, 2, 2, 0, 2,
       2, 0, 0, 0, 0, 2, 1, 2, 1, 2, 0])
```

y_pred_rf_boosted_1

```
array([2, 2, 1, 2, 1, 1, 1, 0, 2, 0, 2, 0, 2, 1, 0, 1, 0, 0, 0, 2, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 2, 0, 2, 2, 0, 1, 0, 0, 2, 0, 0, 0, 1, 0, 2,
       2, 0, 0, 2, 0, 0, 2, 2, 2, 0, 1, 2, 2, 2, 2, 0, 0, 0, 2, 2, 0, 2,
       0, 1, 1, 2, 1, 2, 0, 1, 0, 2, 0, 1, 1, 2, 0, 0, 0, 1, 2, 2, 2, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 2, 1, 2, 0, 1, 1, 0, 2, 2, 2, 0, 0, 0, 2,
       2, 1, 0, 0, 1, 2, 2, 0, 2, 2, 2, 2, 2, 1, 0, 2, 0, 0, 0, 2, 0, 0,
       2, 0, 0, 1, 0, 1, 2, 2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 2, 2, 1, 2, 0, 2, 2, 2, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 2,
       0, 2, 0, 2, 1, 2, 0, 0, 1, 2, 0, 0, 2, 2, 0, 0, 1, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 1, 1, 0, 2, 0, 0, 2, 1, 0, 1,
       1, 0, 2, 2, 0, 0, 2, 0, 2, 0, 0, 0, 0, 2, 1, 1, 1, 2, 2, 0, 0, 0,
       0, 1, 2, 0, 2, 0, 0, 0, 0, 1, 2, 1, 0, 0, 2, 0, 2, 1, 1, 0, 0, 2,
       1, 0, 0, 0, 0, 2, 0, 1, 0, 0, 2, 0, 0, 0, 0, 2, 0, 2, 1, 0, 2,
       0, 0, 2, 0, 0, 2, 2, 0, 0, 2, 2, 0, 1, 0, 1, 0, 0, 1, 2, 0, 0, 0,
       0, 0, 2, 0, 0, 0, 1, 1, 0, 0, 0, 0, 2, 1, 1, 0, 2, 0, 0, 2, 2, 2,
       0, 0, 2, 2, 2, 1, 0, 0, 2, 1, 0, 0, 0, 0, 2, 0, 2, 0, 0, 2, 0, 0,
```

```
            2, 1, 2, 2, 0, 0, 2, 0, 0, 0, 0, 1, 2, 0, 0, 0, 1, 0, 0, 0, 0, 1,
            0, 0, 0, 1, 0, 0, 1, 0, 2, 0, 0, 0, 1, 1, 2, 0, 2, 2, 0, 2, 2, 2,
            2, 1, 0, 1, 0, 2, 1, 0, 2, 1, 2, 0, 2, 0, 0, 0, 2, 1, 1, 0, 0, 1,
            0, 1, 2, 0, 0, 0, 2, 2, 0, 1, 2, 2, 2, 2, 0, 2, 0, 0, 2, 0, 0, 0,
            1, 0, 0, 2, 0, 2, 2, 2, 1, 0, 2, 2, 0, 0, 0, 2, 2, 0, 2, 2, 1, 0,
            0, 0, 0, 1, 0, 0, 2, 2, 0, 0, 0, 1, 1, 0, 2, 2, 0, 2, 2, 2, 0, 2,
            2, 0, 0, 0, 0, 0, 1, 2, 1, 2, 2])
```

```
import pickle

# Create a dictionary to store multiple objects
objects_to_save = [
    X_rem2, X_rem2_train, X_rem2_test,
    y_rem2, y_rem2_train, y_rem2_test,
    myresdf, # dataframe of SVM trials
    mysvm, my_svm_pred, # fitted svm and prediction result
    best_tree_fitted, best_tree_pred, # fitted dec tree and prediction
    best_rf_fitted, y_pred_rf2, # fitted random forest and prediction
    model_rf_boosted2, y_pred_rf_boosted2 # adaboosted random forest and prediction: 82%
]


# Saving all objects in the workspace to a file
with open('241208_objects.pkl', 'wb') as f:
    pickle.dump(objects_to_save, f)

# Download the pickle file
from google.colab import files
files.download('/content/241208_objects.pkl')
```

```
## To re-access the objects:
# Re-upload the pickle file
from google.colab import files
uploaded = files.upload()

# Load the objects from the pickle file
with open('workspace.pkl', 'rb') as f:
    loaded_objects = pickle.load(f)

## Access the loaded objects (they are now in the same order as the original list)
# rf_model_loaded = loaded_objects[0]
# X_rem2_loaded = loaded_objects[1]
```

```
# Metrics
conf_matrix = confusionMatrix(as.factor(pred), as.factor(test_label))
precision = conf_matrix$byClass[, "Precision"]
recall = conf_matrix$byClass[, "Recall"]
f1 = conf_matrix$byClass[, "F1"]
metrics = data.frame(
  Precision = precision,
  Recall = recall,
  F1 = f1,
  AUC = auc_values
)
row.names(metrics) = c("Control","CD","UC")
library(knitr)
kable(metrics, caption = "Evaluation Metrics", digits = 2)
```

```
# ROC curve
for (class in c(0, 1, 2)) {
  # Create binary labels for one-vs-all
  binary_labels = ifelse(test_label == class, 1, 0)

  # Get probabilities for the current class
  class_probs = pred_probs_matrix[, class + 1]

  # Compute ROC and AUC
  roc_curves[[class + 1]] = roc(binary_labels, class_probs)
  auc_values[class + 1] = auc(roc_curves[[class + 1]])
}

# Plot ROC curves for all classes
```

```
# Plot ROC curves for all classes
plot(roc_curves[[1]], col = "red", main = "ROC Curves")
plot(roc_curves[[2]], col = "blue", add = TRUE)
plot(roc_curves[[3]], col = "green", add = TRUE)

# Add a legend
legend("bottomright", legend = c("Control", "CD", "UC"), col = c("red", "blue", "green"), lwd = 2)
```

```
X_rem2_train_l = loaded_objects['X_rem2_train']
X_rem2_test_l = loaded_objects['X_rem2_test']
y_rem2_train_l = loaded_objects['y_rem2_train']
y_rem2_test_l = loaded_objects['y_rem2_test']
```

```
vals_rem2_l = loaded_objects['vals_rem2']
vals_rem2_l # 0: CD, 1: Control, 2: UC
```

```
array(['CD', 'Control', 'UC'], dtype=object)
```

```
mysvm_fitted = loaded_objects['mysvm']
my_svm_pred_l = loaded_objects['my_svm_pred']
best_tree_fitted_l = loaded_objects['best_tree_fitted']
best_tree_pred_l = loaded_objects['best_tree_pred']
best_rf_fitted_l = loaded_objects['best_rf_fitted']
y_pred_rf2_l = loaded_objects['y_pred_rf2']
model_rf_boosted2_l = loaded_objects['model_rf_boosted2']
y_pred_rf_boosted2_l = loaded_objects['y_pred_rf_boosted2']
```

```
#mysvm, my_svm_pred, # fitted svm and prediction result
 #   best_tree_fitted, best_tree_pred, # fitted dec tree and prediction
  #  best_rf_fitted, y_pred_rf2, # fitted random forest and prediction
   # model_rf_boosted2, y_pred_rf_boosted2 # adaboosted random forest and prediction: 82%
```

```
## the factorizing made my labels integers 0,1,2 but one vs rest requires binary labels, only 0/1's
# Binarize the labels for multiclass AUC calculation
y_train_bin = label_binarize(y_rem2_train_l, classes=[0, 1, 2]) # ['CD', 'Control', 'UC']
y_test_bin = label_binarize(y_rem2_test_l, classes=[0, 1, 2]) # ['CD', 'Control', 'UC']
```

```
y_train_bin
```

```
array([[0, 0, 1],
       [0, 0, 1],
       [1, 0, 0],
       ...,
       [0, 1, 0],
       [1, 0, 0],
       [0, 1, 0]])
```

```
y_test_bin
```

```
array([[0, 0, 1],
       [1, 0, 0],
       [0, 1, 0],
       ...,
       [0, 1, 0],
       [0, 0, 1],
       [0, 0, 1]])
```

```
prob_dt = best_tree_fitted_l.predict_proba(X_rem2_test_l)
prob_rf = best_rf_fitted_l.predict_proba(X_rem2_test_l)
prob_rfbd = model_rf_boosted2_l.predict_proba(X_rem2_test_l)
```

```
prob_dt
```

```
array([[0.44444444, 0.        , 0.55555556],
       [1.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        ],
       ...,
       [0.        , 1.        , 0.        ],
       [0.        , 0.        , 1.        ],
       [1.        , 0.        , 0.        ]])
```

```
prob_rf
```

```
array([[0.51424261, 0.00952381, 0.47623358],
       [0.35541934, 0.008     , 0.63658066],
       [0.04544372, 0.92541342, 0.02914286],
       ...,
       [0.13101507, 0.82137021, 0.04761472],
       [0.36229062, 0.00405012, 0.63365927],
       [0.51917625, 0.00689755, 0.4739262 ]])
```

```
prob_rfbd
```

```
array([[0.31312702, 0.25325295, 0.43362003],
       [0.35136819, 0.25545515, 0.39317666],
       [0.26162336, 0.4813651 , 0.25701154],
       ...,
       [0.2672099 , 0.48186319, 0.25092691],
       [0.26737705, 0.24752377, 0.48509918],
       [0.34839084, 0.25536189, 0.39624728]])
```

```
# need to train SVC with probability = TRUE,  the SVM to internally use Platt scaling,
# a method that transforms the decision values into probabilities
### Note: kernels are the mathematical function that transforms data into a linear format so that
### SVMs can classify non-linear data, C controls the balance between maximizing the margin and
### minimizing misclassifications
svcl_15_prob = svm.SVC(kernel="linear", C=1.5, random_state=1, probability = True)

# using my most well performing model with a linear kernel and C=1.5, fit the SVC to my original
# X and y (because previously I conducted the cross_val_score, but had not actually fit it)
# use the fit method to train the specified SVM on the dataset
mysvm_prob = svcl_15_prob.fit(X_rem2_train_l, y_rem2_train_l)

# use the predict method to use the trained model to estimate y_hats for each patient after tmt
mysvm_pred_prob = mysvm_prob.predict(X_rem2_test_l)
```

```
# Get predicted probabilities for all classes
prob_svm = mysvm_prob.predict_proba(X_rem2_test_l)
prob_svm
```

```
array([[4.04507859e-01, 1.54658181e-06, 5.95490594e-01],
       [3.80044686e-01, 2.76674732e-06, 6.19952548e-01],
       [6.08236636e-03, 9.86754259e-01, 7.16337461e-03],
       ...,
       [6.72213465e-03, 9.87732099e-01, 5.54576645e-03],
       [3.23513230e-01, 1.03403874e-05, 6.76476430e-01],
       [1.66543264e-01, 1.27319118e-03, 8.32183545e-01]])
```

```
# double check the accuracy - it's the same
accuracy_score(y_rem2_test_l, mysvm_pred_prob)
```

```
0.7919191919191919
```

```
# Calculate AUC for each class in a one-vs-rest fashion
auc_svm = roc_auc_score(y_test_bin, prob_svm, average='macro', multi_class='ovr')
auc_dt = roc_auc_score(y_test_bin, prob_dt, average='macro', multi_class='ovr')
auc_rf = roc_auc_score(y_test_bin, prob_rf, average='macro', multi_class='ovr')
auc_rfbd = roc_auc_score(y_test_bin, prob_rfbd, average='macro', multi_class='ovr')
```

```
# Calculate AUC for each class in a one-vs-rest fashion
auc_svm
```

```
0.9206967547850082
```

```
auc_dt
```

```
0.8601767929042108
```

```
auc_rf
```

```
0.9329851227083701
```

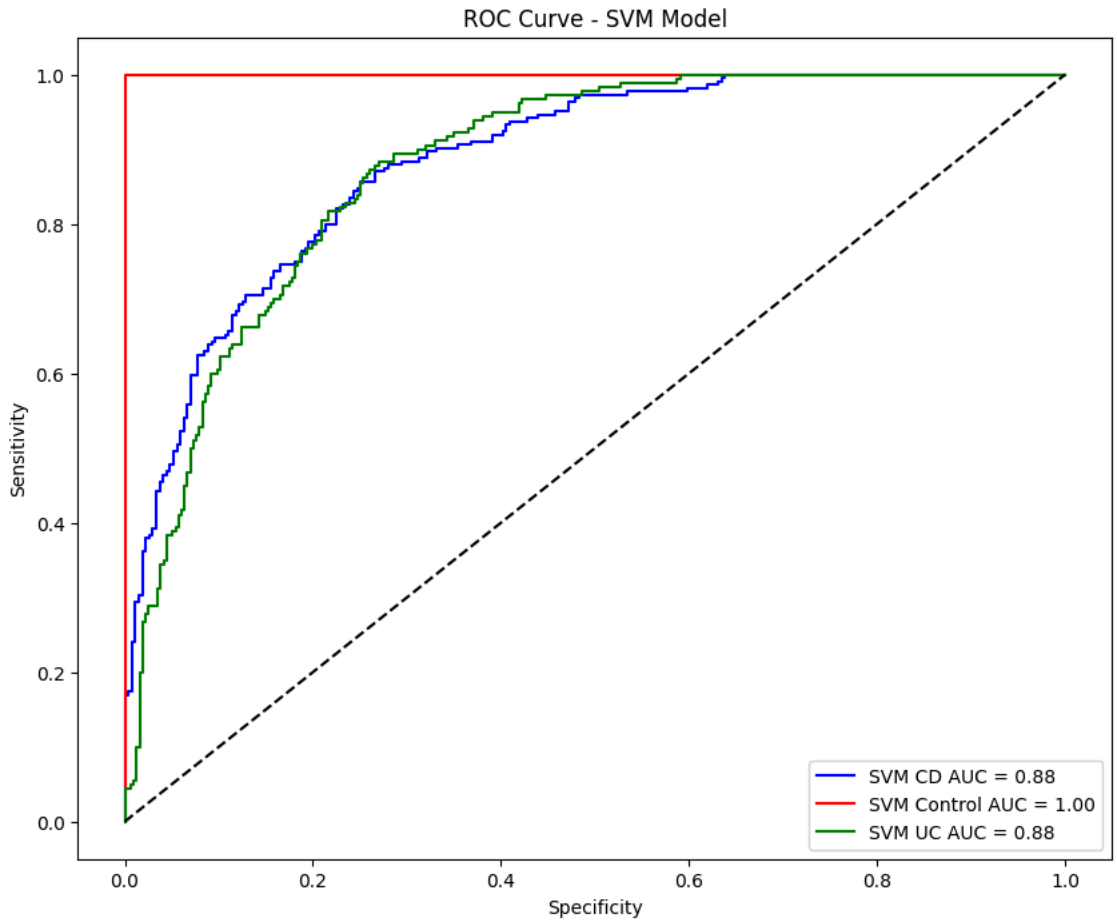
```
auc_rfbd
```

```
0.9371857502966564
```

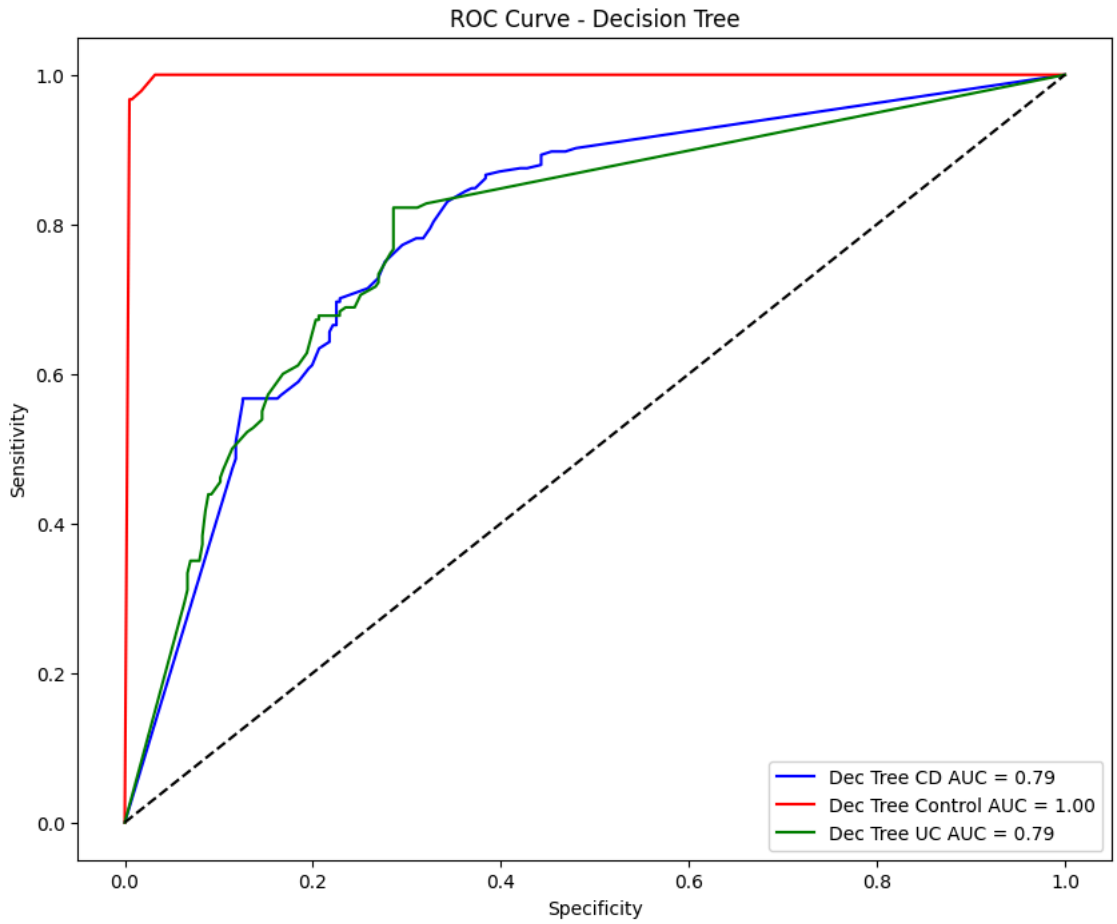
```
for i in range(3):  # 3 classes: 0, 1, 2
    fpr_svm, tpr_svm, _ = roc_curve(y_test_bin[:, i], prob_svm[:, i])

fpr_svm
```

```
array([0.        , 0.        , 0.        , 0.00634921, 0.00634921,
       0.00952381, 0.00952381, 0.01269841, 0.01269841, 0.01587302,
       0.01587302, 0.01904762, 0.01904762, 0.02222222, 0.02222222,
       0.02539683, 0.02539683, 0.03492063, 0.03492063, 0.03809524,
       0.03809524, 0.04126984, 0.04126984, 0.04444444, 0.04444444,
       0.05079365, 0.05079365, 0.05396825, 0.05396825, 0.05714286,
       0.05714286, 0.06031746, 0.06031746, 0.06349206, 0.06349206,
       0.06666667, 0.06666667, 0.06984127, 0.06984127, 0.07301587,
       0.07301587, 0.07619048, 0.07619048, 0.07936508, 0.07936508,
       0.08253968, 0.08253968, 0.08571429, 0.08571429, 0.08888889,
       0.08888889, 0.09206349, 0.09206349, 0.0984127 , 0.0984127 ,
       0.1015873 , 0.1015873 , 0.11111111, 0.11111111, 0.11428571,
       0.11428571, 0.12380952, 0.12380952, 0.14285714, 0.14285714,
       0.14920635, 0.14920635, 0.15238095, 0.15238095, 0.15555556,
       0.15555556, 0.15873016, 0.15873016, 0.16507937, 0.16507937,
       0.16825397, 0.16825397, 0.17460317, 0.17460317, 0.17777778,
       0.17777778, 0.18095238, 0.18095238, 0.18412698, 0.18412698,
       0.18730159, 0.18730159, 0.19365079, 0.19365079, 0.2       ,
       0.2       , 0.2031746 , 0.2031746 , 0.20952381, 0.20952381,
       0.21587302, 0.21587302, 0.22857143, 0.22857143, 0.23492063,
       0.23492063, 0.24444444, 0.24444444, 0.24761905, 0.24761905,
       0.25079365, 0.25079365, 0.25396825, 0.25396825, 0.25714286,
       0.25714286, 0.26031746, 0.26031746, 0.26666667, 0.26666667,
       0.26984127, 0.26984127, 0.28571429, 0.28571429, 0.31111111,
       0.31111111, 0.32063492, 0.32063492, 0.33015873, 0.33015873,
       0.34285714, 0.34285714, 0.34920635, 0.34920635, 0.36507937,
       0.36507937, 0.37142857, 0.37142857, 0.38095238, 0.38095238,
       0.39047619, 0.39047619, 0.41904762, 0.41904762, 0.42222222,
       0.42222222, 0.44761905, 0.44761905, 0.48571429, 0.48571429,
       0.5047619 , 0.5047619 , 0.52698413, 0.52698413, 0.58730159,
       0.58730159, 0.59047619, 0.59047619, 1.        ])
```

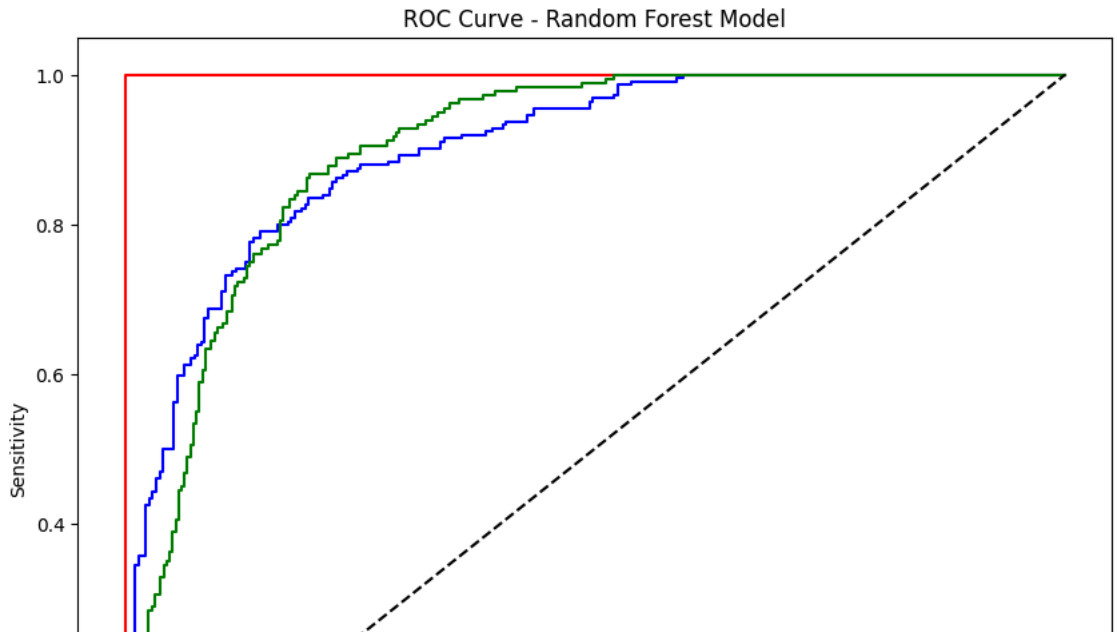
```
# Plot ROC curves for each class
plt.figure(figsize=(10, 8))

# Colors for each class
colors = {0: 'blue', 1: 'red', 2: 'green'}
class_labels = {0: 'CD', 1: 'Control', 2: 'UC'}

# plot the ROC curve for prob_svm
for i in range(3):  # 3 classes: 0, 1, 2
    fpr_svm, tpr_svm, _ = roc_curve(y_test_bin[:, i], prob_svm[:, i])

    # Choose the appropriate color for each class (blue for CD, red for Control, green for UC)
    label = f"SVM {class_labels[i]} AUC = {roc_auc_score(y_test_bin[:, i], prob_svm[:, i]):.2f}"
    plt.plot(fpr_svm, tpr_svm, color=colors[i], linestyle='-', label=label)

# Plot the diagonal line (random classifier)
plt.plot([0, 1], [0, 1], color='black', linestyle='--')

# Customize plot
plt.title('ROC Curve - SVM Model')
plt.xlabel('Specificity')
plt.ylabel('Sensitivity')
plt.legend(loc='lower right')
plt.grid(False)
plt.show()
```




```
# Plot ROC curves for each class
plt.figure(figsize=(10, 8))

# Colors for each class
colors = {0: 'blue', 1: 'red', 2: 'green'}
class_labels = {0: 'CD', 1: 'Control', 2: 'UC'}

# plot the ROC curve for prob_dt
for i in range(3):  # 3 classes: 0, 1, 2
    fpr_dt, tpr_dt, _ = roc_curve(y_test_bin[:, i], prob_dt[:, i])

    # Choose the appropriate color for each class (blue for CD, red for Control, green for UC)
    label = f"Dec Tree {class_labels[i]} AUC = {roc_auc_score(y_test_bin[:, i], prob_dt[:, i]):.2f}"
    plt.plot(fpr_dt, tpr_dt, color=colors[i], linestyle='-', label=label)

# Plot the diagonal line (random classifier)
plt.plot([0, 1], [0, 1], color='black', linestyle='--')

# Customize plot
plt.title('ROC Curve - Decision Tree')
plt.xlabel('Specificity')
plt.ylabel('Sensitivity')
plt.legend(loc='lower right')
plt.grid(False)
plt.show()
```




```
# Plot ROC curves for each class
plt.figure(figsize=(10, 8))

# Colors for each class
colors = {0: 'blue', 1: 'red', 2: 'green'}
class_labels = {0: 'CD', 1: 'Control', 2: 'UC'}

# plot the ROC curve for prob_rf
for i in range(3):  # 3 classes: 0, 1, 2
    fpr_rf, tpr_rf, _ = roc_curve(y_test_bin[:, i], prob_rf[:, i])

    # Choose the appropriate color for each class (blue for CD, red for Control, green for UC)
    label = f"RF {class_labels[i]} AUC = {roc_auc_score(y_test_bin[:, i], prob_rf[:, i]):.2f}"
    plt.plot(fpr_rf, tpr_rf, color=colors[i], linestyle='-', label=label)

# Plot the diagonal line (random classifier)
plt.plot([0, 1], [0, 1], color='black', linestyle='--')

# Customize plot
plt.title('ROC Curve - Random Forest Model')
plt.xlabel('Specificity')
plt.ylabel('Sensitivity')
plt.legend(loc='lower right')
plt.grid(False)
plt.show()
```




```
# Plot ROC curves for each class
plt.figure(figsize=(10, 8))

# Colors for each class
colors = {0: 'blue', 1: 'red', 2: 'green'}
class_labels = {0: 'CD', 1: 'Control', 2: 'UC'}

# plot the ROC curve for prob_rfbd
for i in range(3):  # 3 classes: 0, 1, 2
    fpr_rfbd, tpr_rfbd, _ = roc_curve(y_test_bin[:, i], prob_rfbd[:, i])

    # Choose the appropriate color for each class (blue for CD, red for Control, green for UC)
    label = f"RFB {class_labels[i]} AUC = {roc_auc_score(y_test_bin[:, i], prob_rfbd[:, i]):.2f}"
    plt.plot(fpr_rfbd, tpr_rfbd, color=colors[i], linestyle='-', label=label)

# Plot the diagonal line (random classifier)
plt.plot([0, 1], [0, 1], color='black', linestyle='--')

# Customize plot
plt.title('ROC Curve - Random Forest AdaBoosted')
plt.xlabel('Specificity')
plt.ylabel('Sensitivity')
plt.legend(loc='lower right')
plt.grid(False)
plt.show()
```