**R code (Simulation)**

**1. Species Tree and Gene Trees in Simulation**

library(phybase)

#Simulation 1

tree <- "(((H:0.00402#0.01,C:0.00402#0.01):0.00304#0.01,G:0.00707#0.01):0.00929#0.01,O:0.01635#0.01)#0.01;"

spname <- species.name(tree)

nodematrix <- read.tree.nodes(tree, spname)$nodes

rootnode <- 7

seq <- rep(25, 4)

t1 <- list()

set.seed(100)

for (i in 1:10) {

str <- sim.coaltree.sp(rootnode, nodematrix, 4, seq, name = spname)$gt

t1[[i]] <- str

}

#Simulation 2

tree1 <- "(((H:0.00402#0.001,C:0.00402#0.001):0.00304#0.01,G:0.00707#0.01):0.00929#0.01,O:0.01635#0.01)#0.01;"

spname1 <- species.name(tree1)

nodematrix1 <- read.tree.nodes(tree1, spname1)$nodes

rootnode <- 7

seq <- rep(25, 4)

t2 <- list()

set.seed(100)

for (i in 1:10) {

str1 <- sim.coaltree.sp(rootnode, nodematrix1, 4, seq, name = spname1)$gt

t2[[i]] <- str1

}

# Simulation 3

tree2 <- "(((H:0.00402#0.1,C:0.00402#0.1):0.00304#0.01,G:0.00707#0.01):0.00929#0.01,O:0.01635#0.01)#0.01;"

spname2 <- species.name(tree2)

nodematrix2 <- read.tree.nodes(tree2, spname2)$nodes

rootnode <- 7

seq <- rep(25, 4)

t3 <- list()

set.seed(100)

for (i in 1:10) {

str2 <- sim.coaltree.sp(rootnode, nodematrix2, 4, seq, name = spname2)$gt

t3[[i]] <- str2

}

#Simulation 4

tree3 <- "(((H:0.00402#0.01,C:0.00402#0.01):0.00304#0.001,G:0.00707#0.001):0.00929#0.001,O:0.01635#0.001)#0.001;"

spname3 <- species.name(tree3)

nodematrix3 <- read.tree.nodes(tree3, spname3)$nodes

rootnode <- 7

seq <- rep(25, 4)

t4 <- list()

set.seed(100)

for (i in 1:10) {

str3 <- sim.coaltree.sp(rootnode, nodematrix3, 4, seq, name = spname3)$gt

t4[[i]] <- str3

}

#Simulation 5

tree4 <- "(((H:0.00402#0.01,C:0.00402#0.01):0.00304#0.1,G:0.00707#0.1):0.00929#0.1,O:0.01635#0.1)#0.1;"

spname4 <- species.name(tree4)

nodematrix4 <- read.tree.nodes(tree4, spname4)$nodes

rootnode <- 7

seq <- rep(25, 4)

t5 <- list()

set.seed(100)

for (i in 1:10) {

str4 <- sim.coaltree.sp(rootnode, nodematrix4, 4, seq, name = spname4)$gt

t5[[i]] <- str4

}

write.table(t1,"C:\\Users\\Jian Wu\\Desktop\\STAT thesis\\final\\t1.txt")

write.table(t2,"C:\\Users\\Jian Wu\\Desktop\\STAT thesis\\final\\t2.txt")

write.table(t3,"C:\\Users\\Jian Wu\\Desktop\\STAT thesis\\final\\t3.txt")

write.table(t4,"C:\\Users\\Jian Wu\\Desktop\\STAT thesis\\final\\t4.txt")

write.table(t5,"C:\\Users\\Jian Wu\\Desktop\\STAT thesis\\final\\t5.txt")

**2. Data Pretreatment in Simulation (Treatment 1, the code for other treatments is the same)**

library(phybase)

data <- read.table("output1.txt", header = FALSE, stringsAsFactors = FALSE)

data2 <- data[!grepl("100", data$V1), ]

data2$V1 <- gsub("Cs\\d+", "Cs", data2$V1)

data2$V1 <- gsub("Hs\\d+", "Hs", data2$V1)

data2$V1 <- gsub("Gs\\d+", "Gs", data2$V1)

data2$V1 <- gsub("Os\\d+", "Os", data2$V1)

data2$V2 <- gsub("A", "1000", data2$V2)

data2$V2 <- gsub("C", "0100", data2$V2)

data2$V2 <- gsub("G", "0010", data2$V2)

data2$V2 <- gsub("T", "0001", data2$V2)

nucleotides <- strsplit(data2$V2, "")

nucleotide\_matrix <- do.call(rbind, nucleotides)

result <- data.frame (cbind(data2[, 1], nucleotide\_matrix))

print(result)

data3 <- result[,-1]

library(dplyr)

df <- mutate\_all(data3, as.numeric)

df3 <- data.frame (cbind(data2[, 1], df))

colnames(df3)[colnames(df3) == "data2...1."] <- "Gene"

t11 <- df3 [1:1000,]

t12 <- df3 [1001:2000,]

t13 <- df3 [2001:3000,]

t14 <- df3 [3001:4000,]

t15 <- df3 [4001:5000,]

t16 <- df3 [5001:6000,]

t17 <- df3 [6001:7000,]

t18 <- df3 [7001:8000,]

t19 <- df3 [8001:9000,]

t110 <- df3 [9001:10000,]

final\_datasets\_list <- list()

for (i in 1:10) {

current\_dataset <- get(paste0("t1", i))

OS <- current\_dataset[current\_dataset$Gene == "Os", -1]

HS <- current\_dataset[current\_dataset$Gene == "Hs", -1]

GS <- current\_dataset[current\_dataset$Gene == "Gs", -1]

CS <- current\_dataset[current\_dataset$Gene == "Cs", -1]

chunk\_size <- 25

OS\_list <- list()

HS\_list <- list()

GS\_list <- list()

CS\_list <- list()

for (j in 1:ceiling(nrow(OS) / chunk\_size)) {

start\_row <- (j - 1) \* chunk\_size + 1

end\_row <- min(j \* chunk\_size, nrow(OS))

OS\_chunk <- OS[start\_row:end\_row, ]

OS\_list[[j]] <- OS\_chunk

}

for (j in 1:ceiling(nrow(HS) / chunk\_size)) {

start\_row <- (j - 1) \* chunk\_size + 1

end\_row <- min(j \* chunk\_size, nrow(HS))

HS\_chunk <- HS[start\_row:end\_row, ]

HS\_list[[j]] <- HS\_chunk

}

for (j in 1:ceiling(nrow(GS) / chunk\_size)) {

start\_row <- (j - 1) \* chunk\_size + 1

end\_row <- min(j \* chunk\_size, nrow(GS))

GS\_chunk <- GS[start\_row:end\_row, ]

GS\_list[[j]] <- GS\_chunk

}

for (j in 1:ceiling(nrow(CS) / chunk\_size)) {

start\_row <- (j - 1) \* chunk\_size + 1

end\_row <- min(j \* chunk\_size, nrow(CS))

CS\_chunk <- CS[start\_row:end\_row, ]

CS\_list[[j]] <- CS\_chunk

}

tr\_OS <- do.call(cbind, OS\_list)

tr\_HS <- do.call(cbind, HS\_list)

tr\_GS <- do.call(cbind, GS\_list)

tr\_CS <- do.call(cbind, CS\_list)

colnames(tr\_OS) <- 1:ncol(tr\_OS)

colnames(tr\_HS) <- 1:ncol(tr\_HS)

colnames(tr\_GS) <- 1:ncol(tr\_GS)

colnames(tr\_CS) <- 1:ncol(tr\_CS)

tre\_OS <- data.frame(Gene = paste('Os', collapse=' '), tr\_OS)

tre\_HS <- data.frame(Gene = paste('Hs', collapse=' '), tr\_HS)

tre\_GS <- data.frame(Gene = paste('Gs', collapse=' '), tr\_GS)

tre\_CS <- data.frame(Gene = paste('Cs', collapse=' '), tr\_CS)

treatment <- rbind(tre\_CS, tre\_OS, tre\_HS, tre\_GS)

# Add the final dataset to the list

final\_datasets\_list[[i]] <- treatment

}

final\_dataset\_names <- paste0("treatment1", 1:10)

names(final\_datasets\_list) <- final\_dataset\_names

# Set the ratio for training and test datasets

ratio\_train\_test <- 0.7

set.seed(123)

training\_datasets\_list <- list()

test\_datasets\_list <- list()

for (i in 1:10) {

current\_dataset <- final\_datasets\_list[[i]]

num\_rows <- nrow(current\_dataset)

num\_rows\_train <- round(num\_rows \* ratio\_train\_test)

train\_indices <- sample(1:num\_rows, size = num\_rows\_train, replace = FALSE)

train\_dataset <- current\_dataset[train\_indices, ]

test\_dataset <- current\_dataset[-train\_indices, ]

training\_datasets\_list[[i]] <- train\_dataset

test\_datasets\_list[[i]] <- test\_dataset

}

training\_dataset\_names <- paste0("training\_dataset1", 1:10)

test\_dataset\_names <- paste0("test\_dataset1", 1:10)

names(training\_datasets\_list) <- training\_dataset\_names

names(test\_datasets\_list) <- test\_dataset\_names

# Specify the directory where you want to save the files

output\_directory <- "C:\\Users\\Jian Wu\\Desktop\\STAT thesis\\final\\"

# Loop through each dataset and save the training and test datasets to separate txt files

for (i in 1:10) {

# Get the training and test datasets

training\_dataset <- training\_datasets\_list[[i]]

test\_dataset <- test\_datasets\_list[[i]]

training\_file\_name <- paste0(output\_directory, "training\_dataset", i, ".txt")

test\_file\_name <- paste0(output\_directory, "test\_dataset", i, ".txt")

write.table(training\_dataset, file = training\_file\_name, sep = "\t", quote = FALSE, row.names = FALSE)

write.table(test\_dataset, file = test\_file\_name, sep = "\t", quote = FALSE, row.names = FALSE)

}

**3. Classification Tree (Treatment 1, the code for other treatments is the same)**

library(randomForest)

library(rattle)

library(caret)

library(e1071)

library (tree)

library(rpart)

# Initialize lists to store accuracies for different subsets of columns

tree\_accuracies\_list <- list()

tree\_accuracies1\_list <- list()

# Set the control parameters for the decision tree model

tree\_ctrl <- rpart.control(cp = 0, xval = 10)

# Define the column subsets

column\_subsets <- c(1001, 2001, 5001, 10001)

# Loop through each column subset

for (subset\_size in column\_subsets) {

tree\_accuracies <- numeric(10)

tree\_accuracies1 <- numeric(10)

# Loop through each dataset index

for (i in 1:10) {

# Get the current training and test datasets

current\_training\_dataset <- training\_datasets\_list[[i]]

current\_test\_dataset <- test\_datasets\_list[[i]]

# Select the subset of columns

columns\_to\_select <- 1:subset\_size

current\_training\_dataset <- current\_training\_dataset[, columns\_to\_select]

current\_test\_dataset <- current\_test\_dataset[, columns\_to\_select]

# Build the decision tree model using the training dataset

tree\_optimal <- rpart(Gene ~ ., method = "class",

data = current\_training\_dataset, control = tree\_ctrl)

# Predict the classes for the training dataset and calculate accuracy

tree\_validation <- predict(tree\_optimal, current\_training\_dataset, type = "class")

tree\_accuracies[i] <- mean(tree\_validation == current\_training\_dataset$Gene)

# Predict the classes for the test dataset and calculate accuracy

tree\_validation1 <- predict(tree\_optimal, current\_test\_dataset, type = "class")

tree\_accuracies1[i] <- mean(tree\_validation1 == current\_test\_dataset$Gene)

}

# Store the results in the lists

tree\_accuracies\_list[[as.character(subset\_size)]] <- tree\_accuracies

tree\_accuracies1\_list[[as.character(subset\_size)]] <- tree\_accuracies1

}

# Display the accuracies

for (subset\_size in column\_subsets) {

cat("Training Accuracies (Subset Size =", subset\_size, "):", tree\_accuracies\_list[[as.character(subset\_size)]], "\n")

cat("Test Accuracies (Subset Size =", subset\_size, "):", tree\_accuracies1\_list[[as.character(subset\_size)]], "\n")

}

library(ggplot2)

# Combine accuracies and dataset types into data frames

training\_accuracies\_data <- data.frame(

Subset\_Size = rep(column\_subsets, each = 10),

Accuracy = unlist(tree\_accuracies\_list),

Dataset = rep("Training", each = 40)

)

test\_accuracies\_data <- data.frame(

Subset\_Size = rep(column\_subsets, each = 10),

Accuracy = unlist(tree\_accuracies1\_list),

Dataset = rep("Test", each = 40)

)

# Combine training and test datasets

combined\_accuracies\_data <- rbind(training\_accuracies\_data, test\_accuracies\_data)

# Create a boxplot using ggplot2

ggplot(combined\_accuracies\_data, aes(x = as.factor(Subset\_Size), y = Accuracy, fill = Dataset)) +

geom\_boxplot(position = "dodge") +

labs(title = "Boxplot of Classification Tree Accuracies",

x = "Sequence Length", y = "Accuracy") +

theme\_minimal()+

scale\_x\_discrete(labels = c("250 base pairs", "500 base pairs", "1250 base pairs", "2500 base pairs"))

# Calculate mean accuracies

mean\_accuracies <- tapply(combined\_accuracies\_data$Accuracy, list(combined\_accuracies\_data$Dataset, combined\_accuracies\_data$Subset\_Size), mean)

# Display mean accuracies

cat("Mean Training Accuracies:\n", round(mean\_accuracies["Training", ], 4), "\n")

cat("Mean Test Accuracies:\n", round(mean\_accuracies["Test", ], 4), "\n")

**4. Data Analysis**

library(readxl)

d1 <- read\_xlsx("data.xlsx",sheet = "treatment1")

d2 <- read\_xlsx("data.xlsx",sheet = "treatment2")

d3 <- read\_xlsx("data.xlsx",sheet = "treatment3")

d4 <- read\_xlsx("data.xlsx",sheet = "treatment4")

d5 <- read\_xlsx("data.xlsx",sheet = "treatment5")

d1$Longth <- as.factor(d1$Longth)

d2$Longth <- as.factor(d2$Longth)

d3$Longth <- as.factor(d3$Longth)

d4$Longth <- as.factor(d4$Longth)

d5$Longth <- as.factor(d5$Longth)

library(ggplot2)

# Treatment 1

ggplot(data=d1, mapping=aes(x=Longth, y=Accuracy, fill = MLM)) + geom\_boxplot()+

ggtitle("Boxplot of accuracies under treatment 1") +

xlab("Length") + ylab("Accuracy")

library(dplyr)

library(tidyr)

# Assuming d1 is your dataframe

mean\_table <- d1 %>%

group\_by(Longth, MLM) %>%

summarize(Mean\_Accuracy = mean(Accuracy, na.rm = TRUE)) %>%

pivot\_wider(names\_from = MLM, values\_from = Mean\_Accuracy)

# Print or use mean\_table as needed

print(mean\_table)

# Treatment 2

ggplot(data=d2, mapping=aes(x=Longth, y=Accuracy, fill = MLM)) + geom\_boxplot()+

ggtitle("Boxplot of accuracies under treatment 2") +

xlab("Length") + ylab("Accuracy")

library(dplyr)

library(tidyr)

# Assuming d2 is your dataframe

mean\_table <- d2 %>%

group\_by(Longth, MLM) %>%

summarize(Mean\_Accuracy = mean(Accuracy, na.rm = TRUE)) %>%

pivot\_wider(names\_from = MLM, values\_from = Mean\_Accuracy)

# Print or use mean\_table as needed

print(mean\_table)

# Treatment 3

ggplot(data=d3, mapping=aes(x=Longth, y=Accuracy, fill = MLM)) + geom\_boxplot()+

ggtitle("Boxplot of accuracies under treatment 3") +

xlab("Length") + ylab("Accuracy")

library(dplyr)

library(tidyr)

# Assuming d3 is your dataframe

mean\_table <- d3 %>%

group\_by(Longth, MLM) %>%

summarize(Mean\_Accuracy = mean(Accuracy, na.rm = TRUE)) %>%

pivot\_wider(names\_from = MLM, values\_from = Mean\_Accuracy)

# Print or use mean\_table as needed

print(mean\_table)

# Treatment 4

ggplot(data=d1, mapping=aes(x=Longth, y=Accuracy, fill = MLM)) + geom\_boxplot()+

ggtitle("Boxplot of accuracies under treatment 4") +

xlab("Length") + ylab("Accuracy")

library(dplyr)

library(tidyr)

# Assuming d4 is your dataframe

mean\_table <- d4 %>%

group\_by(Longth, MLM) %>%

summarize(Mean\_Accuracy = mean(Accuracy, na.rm = TRUE)) %>%

pivot\_wider(names\_from = MLM, values\_from = Mean\_Accuracy)

# Print or use mean\_table as needed

print(mean\_table)

# Treatment 5

ggplot(data=d5, mapping=aes(x=Longth, y=Accuracy, fill = MLM)) + geom\_boxplot()+

ggtitle("Boxplot of accuracies under treatment 5") +

xlab("Length") + ylab("Accuracy")

library(dplyr)

library(tidyr)

# Assuming d5 is your dataframe

mean\_table <- d5 %>%

group\_by(Longth, MLM) %>%

summarize(Mean\_Accuracy = mean(Accuracy, na.rm = TRUE)) %>%

pivot\_wider(names\_from = MLM, values\_from = Mean\_Accuracy)

# Print or use mean\_table as needed

print(mean\_table)

**R Code (Empirical)**

**1. Pretreatment (raw data, and updated data’s code are same)**

data2 <- read.table("Raw.txt", header = FALSE, stringsAsFactors = FALSE)

data2$V3[2]

data2$V3 <- gsub("A", "1000", data2$V3)

data2$V3 <- gsub("C", "0100", data2$V3)

data2$V3 <- gsub("G", "0010", data2$V3)

data2$V3 <- gsub("T", "0001", data2$V3)

data2$V3 <- gsub("-", "2222", data2$V3)

data2$V3[2]

# Split the gene sequence into separate columns for each nucleotide

nucleotides <- strsplit(data2$V3, "")

nucleotide\_matrix <- do.call(rbind, nucleotides)

result <- data.frame (cbind(data2[, 2], nucleotide\_matrix))

print(result)

data3 <- result[,-1]

library(dplyr)

data3 <- data3 %>%

mutate\_all(~ replace(., . == 2, 0.5))

df <- mutate\_all(data3, as.numeric)

df3 <- data.frame (cbind(data2[, 2], df))

colnames(df3)[colnames(df3) == "data2...2."] <- "Species"

df3

# Assuming 'your\_data' is your dataframe

set.seed(100) # Set seed for reproducibility

# Generate random indices for the split

index <- sample(1:nrow(df3), 0.7 \* nrow(df3))

# Split the data based on the index

train\_data <- df3[index, ]

test\_data <- df3[-index, ]

train\_data[is.na(train\_data)] <- 0

test\_data[is.na(test\_data)] <- 0

write.table(train\_data,"C:\\Users\\Jian Wu\\Desktop\\STAT thesis\\final\\rawtrain.txt", row.names = FALSE)

write.table(test\_data,"C:\\Users\\Jian Wu\\Desktop\\STAT thesis\\final\\rawtest.txt", row.names = FALSE)

**2. Classification Tree (raw data, and updated data’s code are same)**

library(randomForest)

library(rattle)

library(caret)

library(e1071)

library (tree)

library(rpart)

tree\_ctrl = rpart.control(cp = 0, xval = 10)

tree\_optimal = rpart(Species~., method = "class",

data = train\_data, control = tree\_ctrl)

printcp(tree\_optimal)

plotcp(tree\_optimal)

fancyRpartPlot(tree\_optimal,type=2)

tree\_validation <- predict(tree\_optimal, train\_data, type = "class")

tree\_accuracy <- mean(tree\_validation == train\_data$Species)

tree\_accuracy

tree\_validation1 <- predict(tree\_optimal, test\_data, type = "class")

tree\_accuracy1 <- mean(tree\_validation1 == test\_data$Species)

tree\_accuracy1

**Python Code (Simulation)**

**1. Pretreatment (1 treatment (1 group, other 9 groups is that load other datasets and repeat this data), others is repeated)**

%matplotlib inline

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import scale

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import Ridge, RidgeCV, Lasso, LassoCV

from sklearn.metrics import mean\_squared\_error

import pandas as pd

from sklearn.preprocessing import LabelEncoder

# Load the original training and test datasets

Train = pd.read\_csv('training\_dataset1.txt', delimiter='\s+')

Test = pd.read\_csv('test\_dataset1.txt', delimiter='\s+')

# Create a LabelEncoder object

label\_encoder = LabelEncoder()

# Define subsets of the datasets

subsets = [

Train.iloc[:, :1001], # First 1001 columns

Train.iloc[:, :2501], # First 2501 columns

Train.iloc[:, :5001], # First 5001 columns

Train.iloc[:, :10001], # First 10001 columns

Test.iloc[:, :1001], # First 1001 columns

Test.iloc[:, :2501], # First 2501 columns

Test.iloc[:, :5001], # First 5001 columns

Test.iloc[:, :10001], # First 10001 columns

]

# Initialize an empty list to store datasets

datasets = []

# Loop through each subset

for subset in subsets:

# Create a dataset dictionary

dataset\_dict = {

'X\_train': subset.drop(['Gene'], axis=1).values,

'Y\_train': label\_encoder.fit\_transform(subset['Gene']),

}

# Append the dataset dictionary to the list

datasets.append(dataset\_dict)

# Display the first few rows of the first subset in the list

print(datasets[0]['X\_train'].shape) # Displaying the shape for illustration purposes

print(datasets[0]['Y\_train'][:5]) # Displaying the first 5 labels for illustration purposes

**2. SVM (1 treatment (1 group, other 9 groups is that load other datasets and repeat this data), others is repeated)**

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.svm import SVC

from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder object

label\_encoder = LabelEncoder()

# Define the hyperparameter grid for grid search

tuned\_parameters = [{'C': [0.01, 0.1, 1, 10, 100],

'gamma': [0.5, 1, 2, 3, 4]}]

# Create an SVM classifier with RBF kernel

svm\_classifier = SVC(kernel='rbf')

# Initialize an empty list to store accuracies

accuracies = []

# Define the column lengths for different groupings

column\_lengths = [1001, 2501, 5001, 10001]

for num\_columns in column\_lengths:

# Load and preprocess the dataset

X\_train = Train.iloc[:, :num\_columns].drop(['Gene'], axis=1).values

y\_train = label\_encoder.fit\_transform(Train.iloc[:, :num\_columns]['Gene'])

X\_test = Test.iloc[:, :num\_columns].drop(['Gene'], axis=1).values

y\_test = label\_encoder.fit\_transform(Test.iloc[:, :num\_columns]['Gene'])

# Perform grid search with cross-validation

clf = GridSearchCV(svm\_classifier, tuned\_parameters, cv=10, scoring='accuracy')

clf.fit(X\_train, y\_train)

# Predict labels on the test set

y\_pred = clf.predict(X\_test)

# Calculate accuracy and store it in the list

accuracy = accuracy\_score(y\_test, y\_pred)

accuracies.append(accuracy)

# Display confusion matrix and accuracy for each subset

print(f"\nSubset with {num\_columns} columns:")

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("Accuracy Score:", accuracy)

# Display overall accuracies for each subset

print("\nOverall Accuracies:")

for i, num\_columns in enumerate(column\_lengths):

print(f"Subset with {num\_columns} columns: {accuracies[i]}")

**3. XGboost (1 treatment (1 group, other 9 groups is that load other datasets and repeat this data), others is repeated)**

pip install xgboost

import xgboost as xgb

import numpy as np

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder object

label\_encoder = LabelEncoder()

# Define the hyperparameters for XGBoost

param = {

'max\_depth': 3,

'eta': 0.3,

'verbosity': 1,

'objective': 'multi:softprob',

'num\_class': 4 # specify the number of classes

}

num\_iter = 15

# Define the column lengths for different groupings

column\_lengths = [1001, 2501, 5001, 10001]

# Initialize an empty list to store accuracies

accuracies = []

for num\_columns in column\_lengths:

# Load and preprocess the dataset

X\_train = Train.iloc[:, :num\_columns].drop(['Gene'], axis=1).values

y\_train = label\_encoder.fit\_transform(Train.iloc[:, :num\_columns]['Gene'])

X\_test = Test.iloc[:, :num\_columns].drop(['Gene'], axis=1).values

y\_test = label\_encoder.fit\_transform(Test.iloc[:, :num\_columns]['Gene'])

# Create DMatrix objects

dtrain = xgb.DMatrix(X\_train, label=y\_train, feature\_names=[f'feature\_{i}' for i in range(X\_train.shape[1])])

dtest = xgb.DMatrix(X\_test, label=y\_test, feature\_names=[f'feature\_{i}' for i in range(X\_test.shape[1])])

# Train the model

bst = xgb.train(param, dtrain, num\_iter)

# Make predictions on the test set

preds = bst.predict(dtest)

# Extract best predictions

best\_preds = np.argmax(preds, axis=1)

# Calculate accuracy and store it in the list

accuracy = accuracy\_score(y\_test, best\_preds)

accuracies.append(accuracy)

# Display confusion matrix and accuracy for each subset

print(f"\nSubset with {num\_columns} columns:")

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, best\_preds))

print("Accuracy Score:", accuracy)

# Display overall accuracies for each subset

print("\nOverall Accuracies:")

for i, num\_columns in enumerate(column\_lengths):

print(f"Subset with {num\_columns} columns: {accuracies[i]}")

**4. Catboost (1 treatment (1 group, other 9 groups is that load other datasets and repeat this data), others is repeated)**

pip install catboost

from catboost import Pool, CatBoostClassifier

from sklearn.metrics import accuracy\_score

import numpy as np

# Function to train CatBoost classifier and evaluate accuracy

def train\_and\_evaluate(X\_train, y\_train, X\_test, y\_test, categorical\_features\_indices, num\_columns):

# Ensure categorical\_features\_indices is within bounds

valid\_categorical\_indices = [idx for idx in categorical\_features\_indices if idx < num\_columns]

train\_data = Pool(data=X\_train.iloc[:, :num\_columns], label=y\_train, cat\_features=valid\_categorical\_indices)

valid\_data = Pool(data=X\_test.iloc[:, :num\_columns], label=y\_test, cat\_features=valid\_categorical\_indices)

params = {

'learning\_rate': 0.1,

'eval\_metric': 'Accuracy',

'cat\_features': valid\_categorical\_indices,

'max\_depth': 6,

'early\_stopping\_rounds': 200,

'verbose': 200,

'random\_seed': 42

}

cbc = CatBoostClassifier(\*\*params)

cbc.fit(train\_data, eval\_set=valid\_data, use\_best\_model=True, plot=False)

# Predict labels on the test set

preds = cbc.predict(X\_test.iloc[:, :num\_columns])

# Calculate accuracy

accuracy = accuracy\_score(y\_test, preds)

print(f"Accuracy for {num\_columns} columns: {accuracy}")

return accuracy

# Load and preprocess the dataset

X\_train = Train.drop(['Gene'], axis=1)

y\_train = Train['Gene']

X\_test = Test.drop(['Gene'], axis=1)

y\_test = Test['Gene']

# Create a LabelEncoder object

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# Identify categorical features

categorical\_features\_indices = np.where(X\_train.dtypes != np.float)[0]

# Define the column lengths for different groupings

column\_lengths = [1001, 2501, 5001, 10001]

# Initialize an empty list to store accuracies

accuracies = []

# Loop through each column length

for num\_columns in column\_lengths:

accuracy = train\_and\_evaluate(X\_train, y\_train\_encoded, X\_test, y\_test\_encoded, categorical\_features\_indices, num\_columns)

accuracies.append(accuracy)

# Display overall accuracies for each subset

print("\nOverall Accuracies:")

for i, num\_columns in enumerate(column\_lengths):

print(f"Subset with {num\_columns} columns: {accuracies[i]}")

**5. KNN (1 treatment, others is repeated)**

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

import pandas as pd

# Function to train KNN classifier and evaluate accuracy

def train\_and\_evaluate(X\_train, y\_train, X\_test, y\_test, num\_neighbors, num\_columns):

knn\_model = KNeighborsClassifier(n\_neighbors=num\_neighbors)

knn\_model.fit(X\_train.iloc[:, :num\_columns], y\_train)

# Predict labels on the test set

y\_pred = knn\_model.predict(X\_test.iloc[:, :num\_columns])

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy for {num\_columns} columns: {accuracy}")

return accuracy

# Define the number of neighbors

num\_neighbors = 8

# Define the column lengths for different groupings

column\_lengths = [1001, 2501, 5001, 10001]

# Initialize an empty list to store accuracies

all\_accuracies = []

# Loop through each group of data

for i in range(1, 11):

train\_file = f'training\_dataset{i}.txt'

test\_file = f'test\_dataset{i}.txt'

# Load and preprocess the dataset

Train = pd.read\_csv(train\_file, delimiter='\s+')

Test = pd.read\_csv(test\_file, delimiter='\s+')

X\_train = Train.drop(['Gene'], axis=1)

y\_train = Train['Gene']

X\_test = Test.drop(['Gene'], axis=1)

y\_test = Test['Gene']

# Initialize an empty list to store accuracies for this group

group\_accuracies = []

# Loop through each column length

for num\_columns in column\_lengths:

accuracy = train\_and\_evaluate(X\_train, y\_train, X\_test, y\_test, num\_neighbors, num\_columns)

group\_accuracies.append(accuracy)

all\_accuracies.append(group\_accuracies)

# Display overall accuracies for each subset

print("\nOverall Accuracies:")

for i, group\_accuracy in enumerate(all\_accuracies):

print(f"Group {i+1} Accuracies:")

for j, num\_columns in enumerate(column\_lengths):

print(f"Subset with {num\_columns} columns: {group\_accuracy[j]}")

**6. Neural Network (1 treatment, others are repeated)**

pip install tensorflow

conda install -c conda-forge tensorflow

pip install --upgrade numpy

pip install --upgrade tensorflow numpy

# Find Suitable Epoch

import pandas as pd

from sklearn.preprocessing import LabelEncoder

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import numpy as np

# Create a LabelEncoder object

label\_encoder = LabelEncoder()

# Define the column lengths for different groupings

column\_lengths = [1001, 2501, 5001, 10001]

# Initialize an empty list to store mean accuracies for different epochs and column lengths

mean\_accuracies = []

for num\_columns in column\_lengths:

# Initialize an empty list to store accuracies for this column length

accuracies\_column\_length = []

for group in range(1, 11): # Iterating over 10 groups of data

# Load the original training and test datasets

train\_file = f'training\_dataset{group}.txt'

test\_file = f'test\_dataset{group}.txt'

Train = pd.read\_csv(train\_file, delimiter='\s+')

Test = pd.read\_csv(test\_file, delimiter='\s+')

# Preprocess the training dataset

X\_train = Train.iloc[:, :num\_columns].drop(['Gene'], axis=1).values

y\_train = label\_encoder.fit\_transform(Train.iloc[:, :num\_columns]['Gene'])

# Preprocess the test dataset

X\_test = Test.iloc[:, :num\_columns].drop(['Gene'], axis=1).values

y\_test = label\_encoder.transform(Test.iloc[:, :num\_columns]['Gene']) # Use transform instead of fit\_transform

# Define the neural network model

model = Sequential([

Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)),

Dense(64, activation='relu'),

Dense(len(label\_encoder.classes\_), activation='softmax') # Output layer for classification

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy', # Use this loss for multi-class classification

metrics=['accuracy'])

# Lists to store accuracy values for each epoch

accuracies\_list = []

# Train the model

for epoch in range(41): # Training for 40 epochs

history = model.fit(X\_train, y\_train, epochs=1, batch\_size=32, validation\_split=0.2, verbose=0)

\_, accuracy = model.evaluate(X\_test, y\_test, verbose=0)

accuracies\_list.append(accuracy)

# Append accuracy list after last epoch for this group and column length

accuracies\_column\_length.append(accuracies\_list)

# Calculate the mean accuracy across all groups for this column length

mean\_accuracies.append(np.mean(accuracies\_column\_length, axis=0))

# Now, mean\_accuracies contains the mean accuracy for each column length and each epoch

for column\_index, accuracies in enumerate(mean\_accuracies):

print(f"Column Length {column\_lengths[column\_index]}:")

for epoch\_index, accuracy in enumerate(accuracies):

print(f" Epoch {epoch\_index + 1}: {accuracy}")

import matplotlib.pyplot as plt

# Initialize colors for each column length

colors = ['blue', 'green', 'red', 'purple']

# Plot mean accuracies over epochs for each subset

for i, column\_length in enumerate(column\_lengths):

plt.plot(range(1, 42), mean\_accuracies[i], label=f'{column\_length} base pairs', color=colors[i])

plt.title('Mean Accuracy over Epochs for Different Column Lengths')

plt.xlabel('Epoch')

plt.ylabel('Mean Accuracy')

plt.legend()

plt.grid(True)

# Save the plot as an image file

plt.savefig('mean\_accuracy\_over\_epochs.png', dpi=300, bbox\_inches='tight')

# Show the plot

plt.show()

# Find the accuracy based on the epoch = 20

import pandas as pd

from sklearn.preprocessing import LabelEncoder

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Create a LabelEncoder object

label\_encoder = LabelEncoder()

# Define the column lengths for different groupings

column\_lengths = [1001, 2501, 5001, 10001]

# Initialize an empty list to store accuracies

accuracies = []

for group in range(1, 11): # Iterating over 10 groups of data

train\_file = f'training\_dataset{group}.txt'

test\_file = f'test\_dataset{group}.txt'

# Load the original training and test datasets

Train = pd.read\_csv(train\_file, delimiter='\s+')

Test = pd.read\_csv(test\_file, delimiter='\s+')

group\_accuracies = []

for num\_columns in column\_lengths:

# Preprocess the training dataset

X\_train = Train.iloc[:, :num\_columns].drop(['Gene'], axis=1).values

y\_train = label\_encoder.fit\_transform(Train.iloc[:, :num\_columns]['Gene'])

# Preprocess the test dataset

X\_test = Test.iloc[:, :num\_columns].drop(['Gene'], axis=1).values

y\_test = label\_encoder.transform(Test.iloc[:, :num\_columns]['Gene']) # Use transform instead of fit\_transform

# Define the neural network model

model = Sequential([

Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)),

Dense(64, activation='relu'),

Dense(len(label\_encoder.classes\_), activation='softmax') # Output layer for classification

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy', # Use this loss for multi-class classification

metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_split=0.2, verbose=0)

# Evaluate the model on the test set

\_, accuracy = model.evaluate(X\_test, y\_test)

group\_accuracies.append(accuracy)

# Display accuracy for each subset

print(f"Group {group}, Subset with {num\_columns} columns Accuracy: {accuracy}")

accuracies.append(group\_accuracies)

# Display overall accuracies for each subset

print("\nOverall Accuracies:")

for i, num\_columns in enumerate(column\_lengths):

subset\_accuracies = [group\_accuracies[i] for group\_accuracies in accuracies]

mean\_accuracy = sum(subset\_accuracies) / len(subset\_accuracies)

print(f"Subset with {num\_columns} columns: Mean accuracy: {mean\_accuracy}, Individual accuracies: {subset\_accuracies}")

**7. K means Clustering (1 treatment, others are repeated)**

pip install --user numpy matplotlib

pip install --upgrade threadpoolctl scikit-learn

pip install --user numpy matplotlib

pip install threadpoolctl==3.1.0

# Find the k

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

import pandas as pd

import numpy as np

# Define the column lengths for different groupings and their corresponding base pairs

column\_lengths = [1001, 2501, 5001, 10001]

base\_pairs = [250, 500, 1250, 2500]

num\_clusters = range(1, 11) # Number of clusters to try

# Initialize a dictionary to store inertias for each number of clusters

inertias\_dict = {length: [] for length in column\_lengths}

# Iterate over the column lengths

for length, base\_pair in zip(column\_lengths, base\_pairs):

group\_inertias = []

for group in range(1, 11): # Iterating over 10 groups of data

# Read the train and test datasets

train\_file = f'training\_dataset{group}.txt'

test\_file = f'test\_dataset{group}.txt'

Train1 = pd.read\_csv(train\_file, delimiter='\s+')

Test1 = pd.read\_csv(test\_file, delimiter='\s+')

# Concatenate train and test datasets

Train = pd.concat([Train1, Test1], ignore\_index=True)

# Convert X\_train to a list of tuples for KMeans clustering

X\_train = Train.iloc[:, :length].drop(['Gene'], axis=1).values

data = [tuple(row) for row in X\_train]

# Calculate inertias for different numbers of clusters

inertias = []

for k in num\_clusters:

kmeans = KMeans(n\_clusters=k, random\_state=10)

kmeans.fit(data)

inertias.append(kmeans.inertia\_)

group\_inertias.append(inertias)

# Calculate the average inertia across all groups for each number of clusters

avg\_inertias = np.mean(group\_inertias, axis=0)

# Update the dictionary with average inertias

inertias\_dict[length] = avg\_inertias

# Plot the average inertia over number of clusters for each subset of columns

for length, inertias in inertias\_dict.items():

plt.plot(num\_clusters, inertias, marker='o', label=f'{base\_pairs[column\_lengths.index(length)]} base pairs')

plt.title('Mean Inertia over Number of Clusters')

plt.xlabel('Number of Clusters')

plt.ylabel('Mean Inertia')

plt.xticks(num\_clusters)

plt.legend()

plt.grid(True)

# Save the plot as an image file

plt.savefig('mean\_inertia\_plot.png', dpi=300, bbox\_inches='tight')

# Show the plot

plt.show()

# Find the accuracy based on the k = optimum k

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

import numpy as np

# Function to train KMeans and Logistic Regression models and evaluate accuracy

def train\_and\_evaluate(X\_train, y\_train, X\_test, y\_test, num\_clusters, num\_columns):

# Create and fit KMeans model

kmeans = KMeans(n\_clusters=num\_clusters, random\_state=10).fit(X\_train.iloc[:, :num\_columns])

# Get cluster labels for training and test data

train\_clusters = kmeans.labels\_

test\_clusters = kmeans.predict(X\_test.iloc[:, :num\_columns])

# Apply logistic regression to the cluster assignments

logreg = LogisticRegression(max\_iter=1000, random\_state=10)

logreg.fit(train\_clusters.reshape(-1, 1), y\_train)

# Make predictions on the test set

predictions\_LR = logreg.predict(test\_clusters.reshape(-1, 1))

# Calculate accuracy

accuracy = accuracy\_score(y\_test, predictions\_LR)

print(f"Accuracy for {num\_columns} columns: {accuracy}")

return accuracy

# Define the number of clusters

num\_clusters = 6

# Define the column lengths for different groupings

column\_lengths = [1001, 2501, 5001, 10001]

# Initialize an empty list to store accuracies

accuracies = []

for i in range(1, 11): # Assuming you have 10 groups of data

# Load and preprocess the dataset for each group

train\_file = f'training\_dataset{i}.txt'

test\_file = f'test\_dataset{i}.txt'

X\_train = pd.read\_csv(train\_file, delimiter='\s+').drop(['Gene'], axis=1)

y\_train = pd.read\_csv(train\_file, delimiter='\s+')['Gene']

X\_test = pd.read\_csv(test\_file, delimiter='\s+').drop(['Gene'], axis=1)

y\_test = pd.read\_csv(test\_file, delimiter='\s+')['Gene']

# Loop through each column length

for num\_columns in column\_lengths:

accuracy = train\_and\_evaluate(X\_train, y\_train, X\_test, y\_test, num\_clusters, num\_columns)

accuracies.append(accuracy)

# Display overall accuracies for each subset

print("\nOverall Accuracies:")

group\_num = 1

for i, num\_columns in enumerate(column\_lengths):

print(f"Subset with {num\_columns} columns - Group {group\_num}: {accuracies[i]}")

if (i + 1) % len(column\_lengths) == 0:

group\_num += 1

**Python Code (Simulation)**

**1. Pretreatment (raw data code, and updated data code is same)**

%matplotlib inline

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import scale

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import Ridge, RidgeCV, Lasso, LassoCV

from sklearn.metrics import mean\_squared\_error

train = pd.read\_csv('rawtrain.txt', delimiter='\s+')

test = pd.read\_csv('rawtest.txt', delimiter='\s+')

train.head()

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

y1 = train['Species']

y2 = test['Species']

Y1 = label\_encoder.fit\_transform(y1)

Y2 = label\_encoder.fit\_transform(y2)

Y1

X\_train1 = train.drop(['Species'], axis = 1).values

X\_test1 = test.drop(['Species'], axis = 1).values

y\_train1 = Y1

y\_test1 = Y2

**2. SVM (raw data code, and updated data code is same)**

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import GridSearchCV

tuned\_parameters2 = [{'C': [0.01, 0.1, 1, 10, 100],

'gamma': [0.5, 1,2,3,4]}]

clf2 = GridSearchCV(SVC(kernel='rbf'), tuned\_parameters2, cv=10, scoring='accuracy')

clf2.fit(X\_train1, y\_train1)

clf2.best\_params\_

print(confusion\_matrix(y\_test1, clf2.best\_estimator\_.predict(X\_test1)))

print(clf2.best\_estimator\_.score(X\_test1, y\_test1))

**3. XGboost (raw data code, and updated data code is same)**

pip install xgboost

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import dump\_svmlight\_file

from sklearn.metrics import precision\_score, accuracy\_score

import xgboost as xgb

# Convert your training and testing data to DMatrix format

dtrain1 = xgb.DMatrix(X\_train1, label=y\_train1)

dtest1 = xgb.DMatrix(X\_test1, label=y\_test1)

# Define XGBoost parameters

param = {

'max\_depth': 3,

'eta': 0.3,

'verbosity': 1,

'objective': 'multi:softmax', # Use multi:softmax for multiclass classification

'num\_class': len(np.unique(y\_train1)) # Specify the number of unique classes in your target variable

}

num\_iter = 5 # Number of training iterations

# Train the XGBoost model

bst = xgb.train(param, dtrain1, num\_iter)

# Make predictions on the test data

preds1 = bst.predict(dtest1)

# Extract the best predictions

best\_preds1 = np.asarray([np.argmax(line) for line in preds1])

# Print the predictions and actual labels

print("Predictions:", best\_preds1)

print("Actual labels:", y\_test1)

accuracy1 = accuracy\_score(y\_test1, best\_preds1)

print("Accuracy: %.2f%%" % (accuracy1 \* 100.0))

**4. Catboost (raw data code, and updated data code is same)**

pip install catboost

from catboost import Pool

from catboost import CatBoostClassifier

null\_value\_stats = train.isnull().sum(axis=0)

null\_value\_stats[null\_value\_stats != 0]

X\_train2 = train.drop(['Species'],axis = 1)

y\_train2 = train.Species

X\_test2 = test.drop(['Species'],axis = 1)

y\_test2 = test.Species

train.fillna(-999, inplace=True)

print(X\_train2.dtypes,X\_test2.dtypes)

categorical\_features\_indices = np.where(X\_train2.dtypes != float)[0]

# Step 1: Check for unexpected values in the categorical features

unexpected\_values\_train = X\_train2.astype(str).apply(lambda x: x.str.contains('\.').sum())

unexpected\_values\_test = X\_test2.astype(str).apply(lambda x: x.str.contains('\.').sum())

print("Unexpected values in training set:")

print(unexpected\_values\_train[unexpected\_values\_train != 0])

print("\nUnexpected values in testing set:")

print(unexpected\_values\_test[unexpected\_values\_test != 0])

# Convert all columns to string data type

X\_train2 = X\_train2.astype(str)

X\_test2 = X\_test2.astype(str)

# Re-create Pool objects with the modified data

train\_data1 = Pool(data=X\_train2,

label=y\_train2,

cat\_features=categorical\_features\_indices)

valid\_data1 = Pool(data=X\_test2,

label=y\_test2,

cat\_features=categorical\_features\_indices)

from sklearn.model\_selection import train\_test\_split

%time

params = {

'learning\_rate':0.1,

'eval\_metric':'AUC',

'cat\_features': categorical\_features\_indices, #

'max\_depth':6,

'early\_stopping\_rounds': 200,

'verbose': 200,

'random\_seed': 42

}

cbc\_7 = CatBoostClassifier(\*\*params)

cbc\_7.fit(train\_data1, # instead of X\_train, y\_train

eval\_set=valid\_data1, # instead of (X\_valid, y\_valid)

use\_best\_model=True,

plot=True);

**5. KNN (raw data code, and updated data code is same)**

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

k = 5 # You can choose the appropriate value for k

knn\_model = KNeighborsClassifier(n\_neighbors=k)

knn\_model.fit(X\_train1, y\_train1)

y\_pred = knn\_model.predict(X\_test1)

accuracy = accuracy\_score(y\_test1, y\_pred)

print(f'Accuracy: {accuracy \* 100:.2f}%')

**6. NN (raw data code, and updated data code is same)**

pip install --upgrade tensorflow

pip install --upgrade numpy

# Find Epoch

import pandas as pd

from sklearn.preprocessing import LabelEncoder

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import numpy as np

# Create a LabelEncoder object

label\_encoder = LabelEncoder()

# Load the original training and test datasets

train\_file = 'rawtrain.txt'

test\_file = 'rawtest.txt'

Train = pd.read\_csv(train\_file, delimiter='\s+')

Test = pd.read\_csv(test\_file, delimiter='\s+')

# Preprocess the training dataset

X\_train = Train.drop(['Species'], axis=1).values

y\_train = label\_encoder.fit\_transform(Train['Species'])

# Preprocess the test dataset

X\_test = Test.drop(['Species'], axis=1).values

y\_test = label\_encoder.transform(Test['Species']) # Use transform instead of fit\_transform

# Define the neural network model

model = Sequential([

Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)),

Dense(64, activation='relu'),

Dense(len(label\_encoder.classes\_), activation='softmax') # Output layer for classification

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy', # Use this loss for multi-class classification

metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=40, batch\_size=32, validation\_split=0.2, verbose=1)

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print(f'Test accuracy: {test\_accuracy}')

import matplotlib.pyplot as plt

# Get test accuracy from the history object

test\_accuracy = history.history['accuracy']

# Plot test accuracy

plt.plot(range(1, len(test\_accuracy) + 1), test\_accuracy, label='Test Accuracy')

plt.title('Test Accuracy over Epochs')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.grid(True)

# Save the plot as an image file

plt.savefig('test\_accuracy\_plot.png', dpi=300, bbox\_inches='tight')

# Show the plot

plt.show()

# Find accuracy

import pandas as pd

from sklearn.preprocessing import LabelEncoder

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Load the dataset

train\_file = 'rawtrain.txt'

test\_file = 'rawtest.txt'

# Load the original training and test datasets

Train = pd.read\_csv(train\_file, delimiter='\s+')

Test = pd.read\_csv(test\_file, delimiter='\s+')

# Preprocess the training dataset

X\_train = Train.drop(['Species'], axis=1).values

y\_train = Train['Species']

# Preprocess the test dataset

X\_test = Test.drop(['Species'], axis=1).values

y\_test = Test['Species']

# Apply label encoding to target variables

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

# Define the neural network model

model = Sequential([

Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)),

Dense(64, activation='relu'),

Dense(len(label\_encoder.classes\_), activation='softmax')

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train\_encoded, epochs=10, batch\_size=32, validation\_split=0.2)

# Evaluate the model on the test set

\_, accuracy = model.evaluate(X\_test, y\_test\_encoded)

print(f"Test Accuracy: {accuracy}")

**7. KMeans (raw data code, and updated data code is same)**

# Find K

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

import pandas as pd

# Load the concatenated dataset

train = pd.read\_csv('rawtrain.txt', delimiter='\s+')

test = pd.read\_csv('rawtest.txt', delimiter='\s+')

Train = pd.concat([train, test], ignore\_index=True)

# Prepare the feature matrix

X\_train = Train.drop(['Species'], axis=1).values

# Initialize a KMeans object

kmeans = KMeans()

# Fit the KMeans model and compute the inertia for each number of clusters

inertias = []

for k in range(1, 11):

kmeans.set\_params(n\_clusters=k)

kmeans.fit(X\_train)

inertias.append(kmeans.inertia\_)

# Plot the elbow method

plt.plot(range(1, 11), inertias, marker='o')

plt.title('Elbow Method for Optimal k')

plt.xlabel('Number of Clusters')

plt.ylabel('Inertia')

plt.grid(True)

plt.tight\_layout()

# Save the plot as PNG

plt.savefig('elbow\_plot.png', dpi=300)

# Show the plot

plt.show()

# Find accuracy

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

def load\_data(train\_file, test\_file):

X\_train = pd.read\_csv(train\_file, delimiter='\s+').drop(['Species'], axis=1)

y\_train = pd.read\_csv(train\_file, delimiter='\s+')['Species']

X\_test = pd.read\_csv(test\_file, delimiter='\s+').drop(['Species'], axis=1)

y\_test = pd.read\_csv(test\_file, delimiter='\s+')['Species']

return X\_train, y\_train, X\_test, y\_test

def train\_and\_evaluate(X\_train, y\_train, X\_test, y\_test, num\_clusters):

kmeans = KMeans(n\_clusters=num\_clusters, random\_state=10).fit(X\_train)

train\_clusters = kmeans.labels\_

test\_clusters = kmeans.predict(X\_test)

logreg = LogisticRegression(max\_iter=1000, random\_state=10)

logreg.fit(train\_clusters.reshape(-1, 1), y\_train)

predictions\_LR = logreg.predict(test\_clusters.reshape(-1, 1))

accuracy = accuracy\_score(y\_test, predictions\_LR)

print(f"Accuracy using the whole dataset: {accuracy}")

return accuracy

num\_clusters = 5

train\_file = 'rawtrain.txt'

test\_file = 'rawtest.txt'

X\_train, y\_train, X\_test, y\_test = load\_data(train\_file, test\_file)

accuracy = train\_and\_evaluate(X\_train, y\_train, X\_test, y\_test, num\_clusters)

**Ubuntu Code**

**1. DNA Sequence**

.seq-gen -GTR -l250 t1.txt

**2. Pairwise data**

"/usr/bin/mafft" --auto --inputorder "1.txt" > "data.txt"