***Integrated Single Cell Analysis and Machine Learning for Precision Diagnosis in Human Papillomavirus Associated Cancers***

##############################################################

**Code for Single Cell Analysis**

##############################################################

**STEP1: Install R along with R studio and run the following code in environment**

**#!/usr/bin/env Rscript**

**# R script to install requirements for exercises -------------------------------**

**dir.exists()**

**dir\_exists <- sapply(datadirs, function(dir) dir.exists(dir))**

**print(dir\_exists)**

**## a vector of packages to install (edit in this section) ----------------------**

**### packages could be either on CRAN or bioconductor**

**pkgs <- c("ggplot2", "BiocManager", "sctransform",**

**"devtools", "cowplot", "matrixStats",**

**"ggbeeswarm", "ggnewscale", "msigdbr",**

**"Seurat", "bit64", "scater",**

**"AnnotationDbi",**

**"SingleR", "clusterProfiler", "celldex",**

**"dittoSeq", "DelayedArray",**

**"DelayedMatrixStats",**

**"limma", "SingleCellExperiment",**

**"SummarizedExperiment",**

**"slingshot", "batchelor",**

**"clustree", "edgeR")**

**for (pkg in pkgs) {**

**library(pkg, character.only = TRUE)**

**}**

**install.packages("celldex")**

**BiocManager::install("celldex")**

**library(celldex)**

**install.packages(BiocManager)**

**setwd("C:/Users/Uzair/OneDrive - National University of Sciences & Technology/Desktop/uzair thesis material/CERVICAL CANCER")**

**sampleinfo <- read.csv("sample\_info.csv")**

**datadirs <- file.path(".", sampleinfo$ID)**

**names(datadirs) <- gsub("\_", "-", sampleinfo$ID)**

**datadirs**

**library(Seurat)**

**sparse\_matrix <- Seurat::Read10X(data.dir = datadirs)**

**seu <- Seurat::CreateSeuratObject(counts = sparse\_matrix,**

**project = "Primary",**

**min.cells = 3,**

**min.features = 100)**

**seu**

**View(seu)**

**seu@meta.data**

**metadata\_table <- seu@meta.data**

**write.csv(metadata\_table, "Meta\_Data\_Table.csv", row.names=TRUE)**

**table(seu@active.ident)**

**Cells <- table(seu@active.ident)**

**write.csv(Cells, "Cells.csv", row.names=TRUE)**

**head(seu@meta.data)**

**summary(seu@meta.data$nCount\_RNA)**

**summary(seu@meta.data$nFeature\_RNA)**

**mypalette <- c("#FF0000", "#00796B", "#27CED7", "#FF00CC", "#00FF00","purple","pink","orange","yellow")**

**mypalette2 <- c("#27CED7", "#FF00CC", "#00FF00","#FF2A00", "#00468B","purple","pink","orange","yellow")**

**mypalette3 <- c("#27CED7", "#FF00CC", "#00FF00","#FF2A00", "#00468B","purple","pink","orange","yellow","#FF0000", "#00796B","brown")**

**hist(seu$nCount\_RNA, col = mypalette, main = paste0("Histogram of RNA counts per cell"))**

**hist(seu$nFeature\_RNA, col = mypalette, main = paste0("Histogram of Gene counts per cell"))**

**Seurat::FeatureScatter(seu, feature1 = "nCount\_RNA", feature2 = "nFeature\_RNA", col = mypalette2)**

**Seurat::VlnPlot(seu, features = "nCount\_RNA", cols = mypalette2)**

**Seurat::VlnPlot(seu, features = "nFeature\_RNA", cols = mypalette)**

**Seurat::VlnPlot(seu, features = c("nCount\_RNA",**

**"nFeature\_RNA"))**

**# mitochondrial genes**

**seu <- Seurat::PercentageFeatureSet(seu,**

**pattern = "^MT-",**

**col.name = "percent.mito")**

**# ribosomal genes**

**seu <- Seurat::PercentageFeatureSet(seu,**

**pattern = "^RP[SL]",**

**col.name = "percent.ribo")**

**# hemoglobin genes (but not HBP)**

**seu <- Seurat::PercentageFeatureSet(seu,**

**pattern = "^HB[^(P)]",**

**col.name = "percent.globin")**

**head(seu@meta.data)**

**Seurat::VlnPlot(seu, features = "percent.mito", cols = mypalette2)**

**Seurat::VlnPlot(seu, features = "percent.mito", cols = mypalette2, pt.size = 0)**

**Seurat::VlnPlot(seu, features = "percent.ribo", cols = mypalette2)**

**Seurat::VlnPlot(seu, features = "percent.ribo", cols = mypalette2, pt.size = 0)**

**Seurat::VlnPlot(seu, features = "percent.globin", cols = mypalette2)**

**Seurat::VlnPlot(seu, features = "percent.globin", cols = mypalette2, pt.size = 0)**

**Seurat::VlnPlot(seu, features = c("percent.mito",**

**"percent.ribo",**

**"percent.globin"))**

**Seurat::FeatureScatter(seu,**

**feature1 = "percent.mito",**

**feature2 = "percent.ribo", cols = mypalette2)**

**library(ggplot2)**

**library(Matrix)**

**library(Seurat)**

**most\_expressed\_boxplot <- function(object, ngenes = 20){**

**# matrix of raw counts**

**cts <- Seurat::GetAssayData(object, assay = "RNA", slot = "counts")**

**# get percentage/cell**

**cts <- t(cts)/colSums(cts)\*100**

**medians <- apply(cts, 2, median)**

**# get top n genes**

**most\_expressed <- order(medians, decreasing = T)[ngenes:1]**

**most\_exp\_matrix <- as.matrix((cts[,most\_expressed]))**

**# prepare for plotting**

**most\_exp\_df <- stack(as.data.frame(most\_exp\_matrix))**

**colnames(most\_exp\_df) <- c("perc\_total", "gene")**

**# boxplot with ggplot2**

**boxplot <- ggplot(most\_exp\_df, aes(x=gene, y=perc\_total)) +**

**geom\_boxplot() +**

**coord\_flip()**

**return(boxplot)**

**}**

**most\_expressed\_boxplot(seu, 20)**

**seu <- subset(seu, subset = nFeature\_RNA > 200 &**

**nFeature\_RNA < 5000 &**

**percent.mito < 8)**

**Seurat::VlnPlot(seu, features = c("nFeature\_RNA",**

**"percent.mito"))**

**Seurat::GetAssayData(seu)[1:30,1:30]**

**Assay\_Data\_Before\_Normalization <- Seurat::GetAssayData(seu)[1:30,1:30]**

**write.csv(Assay\_Data\_Before\_Normalization, "Assay\_Data\_Before\_Normalization.csv", row.names=TRUE)**

**############Normalization##############**

**seu <- Seurat::NormalizeData(seu,**

**normalization.method = "LogNormalize",**

**scale.factor = 10000)**

**Seurat::GetAssayData(seu)[1:30,1:30]**

**Assay\_Data\_After\_Normalization <- Seurat::GetAssayData(seu)[1:30,1:30]**

**write.csv(Assay\_Data\_After\_Normalization, "Assay\_Data\_After\_Normalization.csv", row.names=TRUE)**

**#############variable feature############**

**seu <- Seurat::FindVariableFeatures(seu,**

**selection.method = "vst",**

**nfeatures = 2000)**

**# Identify the 10 most highly variable genes**

**top10 <- head(Seurat::VariableFeatures(seu), 10)**

**top10**

**vf\_plot <- Seurat::VariableFeaturePlot(seu)**

**Seurat::LabelPoints(plot = vf\_plot,**

**points = top10, repel = TRUE)**

**############Scaling###########**

**seu <- Seurat::ScaleData(seu,**

**features = rownames(seu))**

**plots <- VlnPlot(seu, features = c("percent.mito",**

**"percent.ribo",**

**"percent.globin"),**

**pt.size =0,**

**combine = FALSE, cols = mypalette2)**

**for(i in 1:length(plots)) {**

**plots[[i]] <- plots[[i]] + geom\_boxplot() + theme(legend.position = 'none')**

**}**

**CombinePlots(plots)**

**##################PCA############**

**seu <- Seurat::RunPCA(seu)**

**Seurat::DimPlot(seu, reduction = "pca")**

**Seurat::DimPlot(seu, reduction = "pca", cols = mypalette2, pt.size = 1)**

**##############Heatmap############**

**Seurat::DimHeatmap(seu, dims = 1:12, cells = 500, balanced = TRUE)**

**Seurat::ElbowPlot(seu, ndims = 40)**

**seu <- Seurat::RunUMAP(seu, dims = 1:25)**

**Seurat::DimPlot(seu, reduction = "umap", cols = mypalette2)**

**# The default number of neighbours is 30. If your dataset is small, a decrease in the number of neighbors can be considered**

**seu <- Seurat::RunUMAP(seu, dims = 1:25, n.neighbors = 30)**

**Seurat::DimPlot(seu, reduction = "umap", cols = mypalette2)**

**# Taking too few PCs we see everything looks connected**

**seu <- Seurat::RunUMAP(seu, dims = 1:5)**

**Seurat::DimPlot(seu, reduction = "umap", cols = mypalette2)**

**#if more precision makes sense, for instance, if the genes that is of interest for your study is not present when the RunPCA was calculated, then an increase in the number of components calculated at start might be interesting to be considered**

**seu <- Seurat::RunUMAP(seu, dims = 1:50)**

**Seurat::DimPlot(seu, reduction = "umap", cols = mypalette2)**

**#changing back to 30 clusters**

**seu <- Seurat::RunUMAP(seu, dims = 1:30)**

**Seurat::DimPlot(seu, reduction = "umap", cols = mypalette2)**

**seu <- Seurat::RunTSNE(seu, dims = 1:30)**

**Seurat::DimPlot(seu, reduction = "tsne", group.by = 'orig.ident', cols = mypalette2)**

**BiocManager::install("harmony")**

**library(Rcpp)**

**seu <- seu %>% RunHarmony('orig.ident', plot\_convergence = F)**

**###############Integration#################**

**library(ggplot2)**

**library(tidyverse)**

**seu@reductions**

**seu.embed <- Embeddings(seu, "harmony")**

**seu.embed**

**############Clustering############**

**library(ggraph)**

**library(clustree)**

**seu\_clusters\_UMAP <- seu %>%**

**RunUMAP(reduction = "harmony", dims = 1:25) %>%**

**FindNeighbors(reduction = "harmony", dims = 1:25) %>%**

**FindClusters(resolution = seq(0.1, 0.8, by=0.1) )**

**seu\_clusters\_TSNE <- seu %>%**

**RunTSNE(reduction = "harmony", dims = 1:20) %>%**

**FindNeighbors(reduction = "harmony", dims = 1:20) %>%**

**FindClusters(resolution = seq(0.1, 0.8, by=0.1) )**

**clustree::clustree(seu\_clusters\_TSNE@meta.data[,grep("RNA\_snn\_res", colnames(seu\_clusters\_TSNE@meta.data))],**

**prefix = "RNA\_snn\_res.")**

**Clusters\_tSNE <- DimPlot(seu\_clusters\_TSNE, reduction = 'tsne', group.by = 'RNA\_snn\_res.0.3', raster = FALSE, cols = mypalette3 )**

**Clusters\_tSNE**

**BiocManager::install("SingleR")**

**detach("package:Matrix", unload = TRUE)**

**library(celldex)**

**library(SingleR)**

**seu\_clusters\_TSNE <- Seurat::SetIdent(seu\_clusters\_TSNE, value = seu\_clusters\_TSNE$RNA\_snn\_res.0.3)**

**Seurat::FeaturePlot(seu\_clusters\_TSNE, reduction = 'tsne', "JCHAIN", label = TRUE)**

**Seurat::FeaturePlot(seu\_clusters\_TSNE, reduction = "tsne", "IGHA1", label = TRUE)**

**Seurat::FeaturePlot(seu\_clusters\_TSNE, reduction = "tsne", "IGHA2", label = TRUE)**

**Seurat::FeaturePlot(seu\_clusters\_TSNE, reduction = "tsne", "S100A9", label = TRUE)**

**Seurat::FeaturePlot(seu\_clusters\_TSNE, reduction = "tsne", "IGHG1", label = TRUE)**

**Seurat::FeaturePlot(seu\_clusters\_TSNE, reduction = "tsne", "IGKV3-20", label = TRUE)**

**Seurat::FeaturePlot(seu\_clusters\_TSNE, reduction = "tsne", "IGHV3-23", label = TRUE)**

**Seurat::FeaturePlot(seu\_clusters\_TSNE, reduction = "tsne", "S100A8", label = TRUE)**

**Seurat::FeaturePlot(seu\_clusters\_TSNE, reduction = "tsne", "TIMP1", label = TRUE)**

**tcell\_genes <- c("IL7R", "LTB", "TRAC", "CD3D")**

**monocyte\_genes <- c("CD14", "CST3", "CD68", "CTSS")**

**Seurat::FeaturePlot(seu\_clusters\_TSNE, reduction = 'tsne', tcell\_genes, ncol=2, label = TRUE)**

**Seurat::VlnPlot(seu\_clusters\_TSNE,**

**features = tcell\_genes,**

**ncol = 2, pt.size = 0, cols = mypalette3)**

**Seurat::VlnPlot(seu\_clusters\_TSNE,**

**features = monocyte\_genes,**

**ncol = 2, pt.size = 0, cols = mypalette3)**

**seu\_clusters\_TSNE <- Seurat::AddModuleScore(seu\_clusters\_TSNE,**

**features = list(tcell\_genes),**

**name = "tcell\_genes")**

**Seurat::FeaturePlot(seu\_clusters\_TSNE, reduction = 'tsne', "tcell\_genes1", label = TRUE)**

**Seurat::VlnPlot(seu\_clusters\_TSNE,**

**"tcell\_genes1",**

**pt.size = 0, cols = mypalette3)**

**mypalette4 <- c("#27CED7", "#FF0000", "#FF9900")**

**s.genes <- Seurat::cc.genes.updated.2019$s.genes**

**g2m.genes <- Seurat::cc.genes.updated.2019$g2m.genes**

**seu\_clusters\_TSNE <- Seurat::CellCycleScoring(seu\_clusters\_TSNE,**

**s.features = s.genes,**

**g2m.features = g2m.genes)**

**Seurat::DimPlot(seu\_clusters\_TSNE, reduction = 'tsne', group.by = "Phase",**

**cols = mypalette4, label = TRUE)**

**ref <- celldex::HumanPrimaryCellAtlasData()**

**class(ref)**

**table(ref$label.main)**

**seu\_SingleR <- SingleR::SingleR(test = Seurat::GetAssayData(seu\_clusters\_TSNE, slot = "data"),**

**ref = ref,**

**BiocManager::install("dittoSeq")**

**SingleR::plotScoreHeatmap(seu\_SingleR)**

**SingleR::plotDeltaDistribution(seu\_SingleR)**

**singleR\_labels <- seu\_SingleR$labels**

**t <- table(singleR\_labels)**

**other <- names(t)[t < 10]**

**singleR\_labels[singleR\_labels %in% other] <- "none"**

**seu\_clusters\_TSNE$SingleR\_annot <- singleR\_labels**

**install.packages(dittoseq)**

**mypalette0 <- c("#00796B", "#27CED7", "#FF00CC", "#00FF00", "#00FFFF", "#FF0000", "#00468B", "#FDAF91", "#5050FF", "#350E20", "#999999", "#7B4173", "#FF9900", "#358000", "#0000CC", "#99CCFF", "#FFCCCC", "#004C00", "#CCFFFF", "#CC99FF", "#9900CC", "#996600", "#666600", "#CCFF00", "#FFCC00", "#000000", "#FF420E", "#79CC3D", "#7E0021", "#FFF7F3", "#6699FF", "#CCCC99")**

**dittoSeq::dittoDimPlot(seu\_clusters\_TSNE, reduction = 'tsne', "SingleR\_annot", size = 0.7)**

**dittoSeq::dittoBarPlot(seu\_clusters\_TSNE, var = "SingleR\_annot", group.by = "orig.ident")**

**dittoSeq::dittoBarPlot(seu\_clusters\_TSNE,**

**var = "SingleR\_annot",**

**group.by = "RNA\_snn\_res.0.3")**

**Seurat::VlnPlot(seu\_clusters\_TSNE,**

**features = "percent.ribo**

**pt.size = 0, cols = mypalette3)**

**#########Differential gene expression###########**

**BiocManager::install("edgeR")**

**library(edgeR)**

**library(limma)**

**de\_genes <- Seurat::FindAllMarkers(seu\_clusters\_TSNE, min.pct = 0.25,**

**only.pos = TRUE)**

**de\_genes <- subset(de\_genes, de\_genes$p\_val\_adj<0.05)**

**write.csv(de\_genes, "de\_genes\_FindAllMarkers.csv", row.names = F, quote = F)**

**write.table(de\_genes, "de\_genes\_FindAllMarkers.txt", row.names = FALSE, quote = FALSE)**

**library(dplyr)**

**top\_specific\_markers <- de\_genes %>%**

**group\_by(cluster) %>%**

**top\_n(3, avg\_log2FC)**

**top\_specific\_markers <- de\_genes %>%**

**group\_by(cluster) %>%**

**top\_n(3, avg\_log2FC)**

**dittoSeq::dittoDotPlot(seu\_clusters\_TSNE, vars = unique(top\_specific\_markers$gene),**

**group.by = "RNA\_snn\_res.0.3")**

**tcell\_genes <- c("IL7R", "LTB", "TRAC", "CD3D")**

**de\_genes[de\_genes$gene %in% tcell\_genes,]**

**seu\_clusters\_TSNE <- Seurat::SetIdent(seu\_clusters\_TSNE, value = "SingleR\_annot")**

**deg\_im\_cells <- Seurat::FindMarkers(seu\_clusters\_TSNE,**

**ident.1 = "Keratinocytes",**

**ident.2 = "Epithelial\_cells",**

**group.by = seu\_clusters\_TSNE$SingleR\_annot,**

**test.use = "wilcox")**

**deg\_im\_cells <- subset(deg\_im\_cells, deg\_im\_cells$p\_val\_adj<0.05)**

**View(deg\_im\_cells)**

**write.csv(deg\_im\_cells, "deg\_im\_cells.csv")**

**############ Matrics File Generation From Single cell analysis################**

**data <- seu@assays$RNA@layers$counts**

**metadata <- seu@meta.data**

**data1 <- (data)**

**str(data1)**

**str(metadata)**

**# Assuming 'data' is a sparse matrix (dgCMatrix) and 'metadata' is a data frame**

**# Check if 'SingleR\_annot' column is present in metadata**

**if ('SingleR\_annot' %in% colnames(metadata)) {**

**# Extract the SingleR\_annot column from metadata**

**singleR\_annot\_values <- metadata$SingleR\_annot**

**# Identify the common row names between metadata and data**

**common\_row\_names <- intersect(rownames(data1), rownames(metadata))**

**# Replace row names in the sparse matrix with values from 'SingleR\_annot'**

**new\_row\_names <- rep("", length(rownames(data1)))**

**new\_row\_names[match(common\_row\_names, rownames(data1))] <- singleR\_annot\_values[match(common\_row\_names, rownames(metadata))]**

**rownames(data1) <- new\_row\_names**

**}**

**# Now 'data' should have row names replaced with values from 'SingleR\_annot' column in 'metadata'**

**install.packages("scrattch.io")**

**remotes::install\_github("AllenInstitute/scrattch.io")**

**devtools::install\_github("AllenInstitute/scrattch.io")**

**library(rhdf5)**

**library(scrattch.io)**

**BiocManager::install("rhdf5")**

**install.packages("rhdf5")**

**str(data1)**

**write\_dgCMatrix\_csv(data1, "MLupdated(data).csv")**

**Code of Machine Learning Models for Cancer Cancer Cell Identification on the base of Gene Expression**

**#############################################**

**Step 3:Install Spyder from the Annaconda and Run That code**

import pandas as pd

import tensorflow as tf

import numpy as np

# Define the paths to the datasets

folder\_path = "E:/ML UZAIR DATASET/"

cer\_dataset\_path = folder\_path + "cer\_dataset1.csv"

hd\_dataset\_path = folder\_path + "HD\_dataset2.csv"

oral\_dataset\_path = folder\_path + "oral\_dataset3.csv"

# Read the datasets

cer\_dataset=pd.read\_csv(cer\_dataset\_path)

hd\_dataset = pd.read\_csv(hd\_dataset\_path)

oral\_dataset = pd.read\_csv(oral\_dataset\_path)

cer\_dataset.columns.values[0] = 'gene'

# Add prefix to the first column where cells are present

cer\_dataset.iloc[:, 0] = cer\_dataset.iloc[:, 0].apply(lambda x: 'D1\_' + str(x) if pd.notnull(x) else x)

hd\_dataset.iloc[:, 0] = hd\_dataset.iloc[:, 0].apply(lambda x: 'D2\_' + str(x) if pd.notnull(x) else x)

oral\_dataset.iloc[:, 0] = oral\_dataset.iloc[:, 0].apply(lambda x: 'D3\_' + str(x) if pd.notnull(x) else x)

# Sort remaining columns alphabetically for each DataFrame

sorted\_cer = cer\_dataset.iloc[:, 1:].sort\_index(axis=1)

sorted\_hd = hd\_dataset.iloc[:, 1:].sort\_index(axis=1)

sorted\_oral = oral\_dataset.iloc[:, 1:].sort\_index(axis=1)

# Merge the extracted first column with the sorted DataFrames

merged\_cer = pd.concat([cer\_dataset.iloc[:, 0], sorted\_cer], axis=1)

merged\_hd = pd.concat([hd\_dataset.iloc[:, 0], sorted\_hd], axis=1)

merged\_oral = pd.concat([oral\_dataset.iloc[:, 0], sorted\_oral], axis=1)

print(merged\_cer)

# Extract common columns

common\_columns = set(merged\_cer.columns[1:]).intersection(merged\_hd.columns[1:], merged\_oral.columns[1:])

common\_columns\_list = list(common\_columns)

# Create new DataFrames containing only the common columns and the first column

common\_cer = merged\_cer[[merged\_cer.columns[0]] + common\_columns\_list]

common\_hd = merged\_hd[[merged\_hd.columns[0]] + common\_columns\_list]

common\_oral = merged\_oral[[merged\_oral.columns[0]] + common\_columns\_list]

print(common\_oral)

# Concatenate row by row along axis=0

merged\_combined = pd.concat([common\_cer, common\_hd, common\_oral], axis=0)

print(merged\_combined)

merged\_combined.to\_csv('merged\_combined.csv', index=False)

**######### SVM model Building and evaluation###########**

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

from sklearn.preprocessing import LabelEncoder

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

# Split the dataset into features (X) and labels (y)

X = merged\_combined.iloc[:, 1:] # Features (all columns except the first one)

y = merged\_combined.iloc[:, 0] # Labels (first column)

# Encode the labels using LabelEncoder

label\_encoder = LabelEncoder()

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

# Split the data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.2, random\_state=42)

# Initialize the SVM classifier

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

# Train the SVM classifier

svm\_classifier.fit(X\_train, y\_train)

# Predict the labels for the test set

y\_pred = svm\_classifier.predict(X\_test)

# Decode the predicted labels

y\_pred\_decoded = label\_encoder.inverse\_transform(y\_pred)

# Calculate the accuracy of the model

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

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1-score:", f1)

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion\_matrix

# Calculate the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Normalize the confusion matrix by row (i.e., by the number of samples in each class)

conf\_matrix\_normalized = conf\_matrix.astype('float') / conf\_matrix.sum(axis=1)[:, np.newaxis]

# Plot the normalized confusion matrix with improved readability

plt.figure(figsize=(25, 25)) # Increase the figure size

ax = sns.heatmap(conf\_matrix\_normalized, annot=True, fmt='.2f', cmap='Blues',

xticklabels=label\_encoder.classes\_, yticklabels=label\_encoder.classes\_,

annot\_kws={"size": 8}, linewidths=.5, linecolor='black') # Adjust font size for annotations

plt.xlabel('Predicted Labels', fontsize=14) # Adjust font size for x-axis label

plt.ylabel('True Labels', fontsize=14) # Adjust font size for y-axis label

plt.title('Normalized Confusion Matrix', fontsize=16) # Adjust font size for title

plt.xticks(fontsize=10, rotation=90) # Rotate x-axis labels for better readability

plt.yticks(fontsize=10) # Adjust font size for y-axis labels

plt.show()

**############### Random Forest Model for Cancer Classification ################**

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

# Initialize the Random Forest classifier

rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the Random Forest classifier

rf\_classifier.fit(X\_train, y\_train)

# Predict the labels for the test set

y\_pred\_rf = rf\_classifier.predict(X\_test)

# Calculate the accuracy of the Random Forest model

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

print("Random Forest Accuracy:", accuracy\_rf)

print("Precision:", precision)

print("Recall:", recall)

print("F1-score:", f1)

# Calculate the confusion matrix

conf\_matrix\_rf = confusion\_matrix(y\_test, y\_pred\_rf)

# Normalize the confusion matrix by row (i.e., by the number of samples in each class)

conf\_matrix\_rf\_normalized = conf\_matrix\_rf.astype('float') / conf\_matrix\_rf.sum(axis=1)[:, np.newaxis]

# Plot the normalized confusion matrix with improved readability

plt.figure(figsize=(25, 25)) # Increase the figure size

ax = sns.heatmap(conf\_matrix\_rf\_normalized, annot=True, fmt='.2f', cmap='Blues',

xticklabels=label\_encoder.classes\_, yticklabels=label\_encoder.classes\_,

annot\_kws={"size": 8}, linewidths=.5, linecolor='black') # Adjust font size for annotations

plt.xlabel('Predicted Labels', fontsize=14) # Adjust font size for x-axis label

plt.ylabel('True Labels', fontsize=14) # Adjust font size for y-axis label

plt.title('Normalized Confusion Matrix for Random Forest Model', fontsize=16) # Adjust font size for title

plt.xticks(fontsize=10, rotation=90) # Rotate x-axis labels for better readability

plt.yticks(fontsize=10) # Adjust font size for y-axis labels

plt.show()

**# Artifical Neural Network model Build and evaluation for Cell identification on the base of Gene Expression**

# ching the layers

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

from keras.models import Sequential

from keras.layers import Dense, Dropout

from keras.utils import to\_categorical

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import classification\_report

# Features and Labels

X = merged\_combined.iloc[:, 1:].values # Features

y = merged\_combined.iloc[:, 0].values # Labels

# Convert labels to categorical

label\_encoder = LabelEncoder()

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

y\_categorical = to\_categorical(y\_encoded)

# Number of classes

num\_classes = len(label\_encoder.classes\_)

# Split the data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_categorical, test\_size=0.2, random\_state=42)

# Define the model architecture with modifications

model = Sequential()

model.add(Dense(128, activation='relu', input\_dim=X\_train.shape[1])) # Increase the number of neurons

model.add(Dropout(0.2)) # Add Dropout for regularization

model.add(Dense(64, activation='relu')) # Add another hidden layer

model.add(Dense(num\_classes, activation='softmax')) # Output layer

# Compile the model with a different optimizer and learning rate

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model with a different number of epochs

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_test, y\_test))

# Evaluate the model

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

print("Test Loss:", loss)

print("Test Accuracy:", accuracy)

# Predictions on test set

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

# Convert predictions from categorical to labels

y\_pred\_labels = label\_encoder.inverse\_transform(y\_pred.argmax(axis=1))

y\_test\_labels = label\_encoder.inverse\_transform(y\_test.argmax(axis=1))

# Generate classification report

report = classification\_report(y\_test\_labels, y\_pred\_labels)

print(report)

import matplotlib.pyplot as plt

# Plot training loss and validation loss

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Training and Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()

# Plot training accuracy and validation accuracy

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Training and Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

**# Xgboost Model building and implementation for cancer cells identification on the base of gene expression**

import xgboost as xgb

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

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

import seaborn as sns

# Assuming 'merged\_combined' is already prepared

# Split X (features) and y (target)

X = merged\_combined.iloc[:, 1:] # All columns except the first one

y = merged\_combined.iloc[:, 0] # First column

# Convert categorical labels to numeric labels

label\_encoder = LabelEncoder()

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

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.3, random\_state=42)

# Create the XGBoost classifier

xgb\_clf = xgb.XGBClassifier(use\_label\_encoder=False, eval\_metric='mlogloss')

# Train the model

xgb\_clf.fit(X\_train, y\_train)

# Make predictions

y\_pred = xgb\_clf.predict(X\_test)

# Convert numeric predictions back to original labels

y\_pred\_labels = label\_encoder.inverse\_transform(y\_pred)

y\_test\_labels = label\_encoder.inverse\_transform(y\_test)

# Evaluate the model

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

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

# Confusion Matrix

conf\_matrix = confusion\_matrix(y\_test\_labels, y\_pred\_labels)

print("Confusion Matrix:")

print(conf\_matrix)

# Plot Confusion Matrix

plt.figure(figsize=(10, 8))

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=label\_encoder.classes\_, yticklabels=label\_encoder.classes\_)

plt.title("Confusion Matrix")

plt.xlabel("Predicted Labels")

plt.ylabel("True Labels")

plt.show()

# Classification Report

print("Classification Report:")

class\_report = classification\_report(y\_test\_labels, y\_pred\_labels)

print(class\_report)