Utils

2023-04-30

```
get_ordered_files <- function(img_dir, prefix) {

    # filtering by prefix and suffix
    files_gm <- list.files(path = img_dir, pattern = paste0("^", prefix, ".*gm.nii.gz"),
                           recursive = FALSE)
    files_wm <- list.files(path = img_dir, pattern = paste0("^", prefix, ".*wm.nii.gz"),
                           recursive = FALSE)

    # adding on full path to image
    files_gm <- paste0(img_dir, "/", files_gm)
    files_wm <- paste0(img_dir, "/", files_wm)

    # sort the files
    sorted_gm_files <- sort(files_gm)
    sorted_wm_files <- sort(files_wm)

    return(list(sorted_gm_files, sorted_wm_files))
}




create_design_mat <- function(metadata, file_ls, wm_files=NULL, gm_files=NULL, comb=FALSE) {

    # create empty data frames with required dimensions
    n_rows <- length(file_ls)
    n_cols <- prod(dim(readnii(file_ls[1])))
    X_data <- data.frame(matrix(nrow = n_rows, ncol = n_cols))
    X_info <- data.frame(matrix(nrow = n_rows, ncol = 2))
    colnames(X_data) <- paste0("V", 1:n_cols)
    colnames(X_info) <- c("Subject", "Image_ID")

    for (i in 1:length(file_ls)){

        if (isTRUE(comb)) {

            wm_img <- readnii(wm_files[i])
            gm_img <- readnii(gm_files[i])
            img <- wm_img
            img[which(gm_img[] != 0)] <- gm_img[which(gm_img[] != 0)]

            } else {

            img <- readnii(file_ls[i])
        }
```

```
        split_str <- unlist(strsplit(file_ls[i], split='-'))[2:3]
        sub_ID <- split_str[1]
        img_ID <- split_str[2]

        data <- as.vector(img)

        X_data[i,] <- data
        X_info[i,] <- c(sub_ID, img_ID)

    }

    df <- cbind(X_info, X_data)

    df$Image_ID <- factor(df$Image_ID)
    df$Subject <- factor(df$Subject)

    # combining metadata with image matrix
    joined_df <- merge(df, metadata, by = c("Image_ID", "Subject"), all.x=TRUE)

    # select only complete cases and relevant columns

    joined_df <- joined_df[complete.cases(joined_df),] %>%
    select(-c(Image_ID, Subject))

    # removing 0 variance columsn because it will interfer with PCA
    y <- joined_df$Group
    X <- joined_df[, !names(joined_df) %in% c("Group")]

    # freqCut is the ratio of the most common variable to the second most common var
    # so if there are 435 values, if the ratio of the most common value to the second most common
    # value is greater than 420 to 15, then it will proceed to the next step
    # uniqueCut will say that if the above is true, of the 435 values, if there are less than 10
    # unique values, it will then remove the column
    # it needs to fulfill both requirements.
    # the function will return the columns that need fit the requirements
    # experiment with other vals
    nzv_cols <- nearZeroVar(X, freqCut = 420/15, uniqueCut = 10,
                            saveMetrics = FALSE,
                            foreach = TRUE,
                            allowParallel=TRUE)

    # when nzv_cols is null, subsetting by null will return a null
    if(length(nzv_cols)!=0) {
        X_nzv <- X[,-nzv_cols]
    } else {
        X_nzv <- X
    }

    return(list(X=X_nzv, y=y, nzv_cols=nzv_cols))
}
```

```r
train_test_split_proportional <- function(y, split_prop, set_seed=FALSE) {
    N <- length(y)

    ### CN 151, MCI 206, AD 77, calculated from metadata
    # find proportion of n_i from N
    cn_prop <- round(151/N, 3)
    mci_prop <- round(206/N, 3)
    ad_prop <- round(77/N, 3)

    # calculate num of train samples from given split
    train_n <- round(split_prop * N)

    # calc proportion size of each group from train sample
    train_cn_n <- round(train_n * cn_prop)
    train_mci_n <- round(train_n * mci_prop)
    train_ad_n <- round(train_n * ad_prop)

    train_indices <- rep(FALSE, N)

    if(set_seed) {set.seed(10)}
    # grab indices for group
    train_indices[sample(which(y == 0), train_cn_n)] <- TRUE
    train_indices[sample(which(y == 1), train_mci_n)] <- TRUE
    train_indices[sample(which(y == 2), train_ad_n)] <- TRUE

    return(train_indices)
}




perform_pca <- function(nzv) {

    y <- nzv$y
    X <- nzv$X

    train_idx <- train_test_split_proportional(y, 0.8, set_seed=TRUE)

    X_train <- X[train_idx, ]
    X_test <- X[!train_idx, ]
    y_train <- y[train_idx]
    y_test <- y[!train_idx]

    # Preprocess and scale the data
    scaler <- scale(X_train, center=TRUE, scale=TRUE)
    X_train_scaled <- scaler

    # apply the scale we did to the train to the test
    X_test_scaled <- scale(X_test,
                           center = attr(scaler, "scaled:center"),
                           scale = attr(scaler, "scaled:scale"))
```

```
    # fit and transform the PCA model on the training set
    pca <- prcomp(X_train_scaled, center = TRUE, scale. = TRUE)

    # summary to calculate variance
    sum_pca <- summary(pca)
    #calculate total variance explained by each principal component
    df_pca <- data.frame(t(sum_pca$importance))
    prop_var <- df_pca$Proportion.of.Variance

    # predict applied the pca to the data, in this case to the train
    X_train_pca <- as.data.frame(pca$x)

    # calculate cumulative variance
    cumulative_variance <- cumsum(pca$sdev^2 / sum(pca$sdev^2))

    # find the index where cumulative variance reaches 95%
    n_components <- which(cumulative_variance >= 0.95)[1]
    print(paste("n components:", n_components))

    sdev <- pca$sdev

    # Transform the test set using the trained PCA model
    X_test_pca <- as.data.frame(predict(pca, newdata=X_test_scaled))

    return(list(X_train_pca=X_train_pca[,1:n_components], y_train=y_train,
                X_test_pca=X_test_pca[,1:n_components], y_test=y_test,
                cum_var=cumulative_variance, prop_var=prop_var,
                sdev=sdev, X_train=X_train))
}




calc_loading_vec <- function(pca_ls, n_components) {
    #' Description:
    #' Calculates the normalized coefficients of the principal components of a training dataset using P

    #' Outputs:
    #' - coef_vec: A vector of normalized coefficients representing the contribution of each original p
    #'       intensity to the respective principal component. The coefficients are scaled to values bet

    #' Details:
    #' - The loading vectors or coefficients of the principal components indicate the contribution
    #'      of each original pixel intensity to the respective principal component.
    #' - To calculate the loading vectors or coefficients, we invert the PCA transformation by multiply
    #'      the pseudo-inverse of the original pixel intensity matrix with the principal component matr
    #' - The singular values or PC weights are multiplied by the loading vectors or coefficients to get
    #' - The resulting vector can be used to generate a new image that highlights the regions
    #'      of the brain that contribute the most to the principal components.


    # Ax = b -> x = A^{-1}b
```

```
    A_pinv <- pinv(as.matrix(pca_ls$X_train))
    b <- as.matrix(pca_ls$X_train_pca)
    # x is the projection matrix
    x <- A_pinv %*% b

    # we only care about the magnitude of the loading vector. it can be positive or negative and will i
    x <- abs(x)

    # weight
    w <- as.matrix(pca_ls$sdev[1:n_components]^2)

    # normalized coefficient vector
    coef_vec <- x %*% w

    return(coef_vec)
}



find_contibuting_regions <- function(nzv_cols, loading_vec, slice) {
    N_pixels <- 121 * 145 * slice

    img_vec <- rep(NA, N_pixels)

    img_vec[nzv_cols] <- 0

    img_vec[-which(img_vec==0)] <- loading_vec

    # intensity values are scaled between 0 and 1
    img_vec <- (img_vec - min(img_vec)) / (max(img_vec) - min(img_vec))

    # img_intensities <- array(img_vec, dim = c(121, 145, slice))

    return(as.vector(img_vec))
    }




plot_axial_lv <- function(loading_mat) {
    # Define the range of slices to plot
    slices_to_plot <- seq(from = 1, to = dim(loading_mat)[1], by = 5)

    # Set up the plotting window
    par(mfrow = c(5,5), mar = c(0, 0, 0, 0))

    # Create a loop to plot each selected slice
    for (i in 1:length(slices_to_plot)) {
      # Extract the current slice from the data
      current_slice <- loading_mat[slices_to_plot[i], , ]

      # Add slice number and sum of slice as annotations in the upper right corner
```

```
      annotation <- paste0(slices_to_plot[i], ":", round(sum(current_slice)))

      # Plot the current slice on the sagittal plane with annotations
      image(current_slice, col = gray(0:255/255), axes = FALSE, xlab = "", ylab = "")
      text(x = .8, y = 0.9,
           labels = annotation,col='white')
    }
}



plot_coronal_lv <- function(loading_mat) {
    # Define the range of slices to plot
    slices_to_plot <- seq(from = 1, to = dim(loading_mat)[2], by = 6)

    # Set up the plotting window
    par(mfrow = c(5,5), mar = c(0, 0, 0, 0))

    # Create a loop to plot each selected slice
    for (i in 1:length(slices_to_plot)) {
      # Extract the current slice from the data
      current_slice <- loading_mat[,slices_to_plot[i] , ]

      # Add slice number and sum of slice as annotations in the upper right corner
      annotation <- paste0(slices_to_plot[i], ":", round(sum(current_slice)))

      # Plot the current slice on the sagittal plane with annotations
      image(current_slice, col = gray(0:255/255), axes = FALSE, xlab = "", ylab = "")
      text(x = .8, y = 0.9,
           labels = annotation,col='white')
    }
}







plot_cumul_var <- function(cum_var, prefix) {

  n <- length(cum_var)

  df <- data.frame(cumulative_variance=cum_var, principal_component=1:n)
  pca_n <- which(cum_var >= 0.95)[1]

  p <- ggplot(df, aes( x = principal_component, y = cum_var)) +
    geom_line() +
    xlim(0,n) +
    scale_x_continuous(breaks = seq(0, n, by = 20)) +
    geom_vline(aes(xintercept = pca_n, linetype = paste0("ncomponents: ", pca_n)), col = 'red', show.leg
    geom_hline(aes(yintercept = 0.95, linetype = paste0("95% sigma")), col = 'blue', show.legend = TRUE)
```

```
    theme_minimal() +
    xlab( 'Principal Component Number' ) +
    ylab( 'Cumulative Explained Variance' ) +
    ggtitle(paste0('Cumulative Explained Variance ', prefix)) +
    scale_linetype_manual(name = "Line Types", values = c(2, 1),
                      guide = guide_legend(override.aes = list(color = c("blue", "red"))))

  return(p)
}




plot_pc1_pc2 <- function(X_train, y_train, X_test, y_test, prop_var, prefix) {

    pca_train_2d <- data.frame(X_train) %>%
        select(c('PC1','PC2')) %>%
        mutate(labels = as.factor(y_train), data = "Train Data")

    cat_mean <- pca_train_2d %>%
        group_by(labels) %>%
        summarise( PC1_mean = mean( PC1 ),
                   PC2_mean = mean( PC2 ) )

    p <- ggplot( pca_train_2d, aes( x = PC1, y = PC2, color = labels)) +
        geom_point(size = 1, alpha=0.7) +
        theme_classic() +
        geom_text(data=cat_mean, aes( x = PC1_mean, y = PC2_mean, label =labels),
                  color = 'black', size = 5 ) +
        guides(colour = guide_legend(override.aes = list(size=10))) +
        xlab(paste0("PC1:", prop_var[1]*100, "%")) +
        ylab(paste0("PC2:", prop_var[2]*100, "%")) +
        ggtitle(paste0('Data Projections onto PC1 & PC2 feature space ', prefix) )

  return(p)
}




calc_scores <- function(model, pca_ls, mod='mn', sim=FALSE) {

    if (sim) {
        N <- 87
        real_indices <- rep(FALSE, N)
        y <- pca_ls$y_test
        # https://www.alz.org/media/Documents/alzheimers-facts-and-figures-special-report-2022.pdf
        # paper said 12-18% of people ages > 60 have MCI
        # 10% of people ages > 60 have AD
        n <- sum(y==0)
        mci_n <- 5 # taking 14% from 37
        ad_n <- 2 # 3% from 37
        cn_n <- (y==0) # 30 samples
```

```
    # randomly sample indices for group
    set.seed(10)
    real_indices[sample(which(y == 2), ad_n)] <- TRUE
    real_indices[sample(which(y == 1), mci_n)] <- TRUE
    real_indices[cn_n] <- TRUE

    y_test <- pca_ls$y_test[real_indices]
    x_test <- pca_ls$X_test_pca[real_indices,]
} else {
    y_test <- pca_ls$y_test
    x_test <- pca_ls$X_test_pca
    }


if (mod=='mn') {
        # predict on the test set
        y_preds <- predict(model, newx = as.matrix(x_test), s = "lambda.min", type = "class")

    } else if (mod=='lda') {
        y_preds <- predict(model, x_test, type = 'class')
        y_preds <- y_preds$class
}

# Calculate confusion matrix
sumry <- confusionMatrix(data=factor(y_preds,levels=c(0,1,2)),
                         reference=factor(y_test,levels=c(0,1,2)),
                         dnn=c("Prediction", "Actual"))
cm <- sumry$table

class_scores <- list()

for (i in 1:3) {
    tp <- cm[i, i]
    fn <- sum(cm[,i]) - tp
    fp <- sum(cm[i, ]) - tp

    precision <- tp / (tp + fp)
    recall <- tp / (tp + fn)
    f1 <- (2*precision*recall)/(precision+recall)

    class_scores[[i]] <- list(precision = precision, recall = recall, f1 = f1, tp= tp, fn = fn, fp =
}

# calculates sum of diagnols of the correctly classified
true <- sum(diag(cm))

# calculates sum of off diagnols
false <- sum(cm - diag(diag(cm)))

# Calculate accuracy
acc <- (true) / (false + true)
mce <- 1-acc
```

```
    # add accuracies to scoring values
    acc_scores <- list(acc=acc,mce=mce)
    class_scores[[4]] <- acc_scores


    ### Cohens Kappa
    raters <- cbind(factor(y_test,levels=c(0,1,2)),
                    factor(y_preds,levels=c(0,1,2)))

    # calculate cohens kappa
    kappa <- cohen.kappa(x=raters)

    # unweighted kappa values
    var_kappa <- kappa$var.kappa
    lw_bound <- kappa$confid[1,1]
    up_bound <- kappa$confid[1,3]
    est_kappa <- kappa$confid[1,2]

    kappa_scores <- list(var = var_kappa, lb = lw_bound, ub = up_bound, est = est_kappa)

    # add to rest of scoring values
    class_scores[[5]] <- kappa_scores


    return(list(mod=model, scores=class_scores, cm=cm, kappa=kappa))
}



get_other_scores_table <- function(scores) {
    # example list of lists
    names(scores) <- c("CN", "MCI", "AD")

    # convert to dataframe
    df <- data.frame(
      precision = sapply(scores[1:3], "[[", 1),
      recall = sapply(scores[1:3], "[[", 2),
      f1 = sapply(scores[1:3], "[[", 3)
    )

    return(t(df))
    }




balanced_weights <- function() {
        N <- 434
        n_classes <- 3

        cn_w <- N / (n_classes * 151)
        mci_w <- N / (n_classes * 206)
        ad_w <- N / (n_classes * 77)
```

```
        return(c(cn_w, mci_w, ad_w))
    }


mn_reg <- function(pca_ls,weight=TRUE) {

    # y are classes 0,1,2 so if I add 1 = 1,2,3
    # i can then specify which index weight balancedweights and add to new vector
    if (weight) {
        y_weights <- balanced_weights()[pca_ls$y_train+1]
        } else {
        y_weights <- NULL
        }




    # base no cv, just wanted to plot the loss on
    fit <- glmnet(as.matrix(pca_ls$X_train_pca), pca_ls$y_train,
              family = "multinomial",
                     alpha = 0.5, # 1 lasso/l1, 0 ridge/l2, 0.5 elastic net
                     type.measure='deviance',
                     type.multinomial='grouped',
                    weights=y_weights)

    # plot the l1 convergence
    plot(fit, xvar = "lambda", label = TRUE)



    # fit a penalized multinomial logistic regression model
    # alpha = 1 is L1, 0 is L2
    clf <- cv.glmnet(as.matrix(pca_ls$X_train_pca), pca_ls$y_train,
                         nfolds = 10,
                         family = "multinomial",
                         type.measure='deviance',
                         alpha = 0.5,
                        weights=y_weights,
                    type.multinomial = "grouped")
    # non grouped lasso each feature is independent
    # in the case of mri data, when mapped to 1 dimension the voxels would have dependence and so shoul

    # here we can see the cross validated fit for each log(lambda)
    # you can see the upper and lower standard deviations with the points
    # the first line is the lambda min that gives the minimum mean cross-validated misclassificaiton er
    # the one to the right  is the value of lambda that gives the most regularized model such
    # that the cross-validated error is within one standard error of the minimum.


    plot(clf)

  return(calc_scores(clf, pca_ls, mod='mn', sim=FALSE))
}
```

```r
lda_reg <- function(pca_ls) {

  # Fit logistic regression model
  clf <- lda(pca_ls$y_train ~ ., data = pca_ls$X_train_pca)

  return(calc_scores(clf, pca_ls, mod='lda', sim=FALSE))
}
```

```r
summary_stats <- function(model) {
    summary_mn <- summary(model)
    coefs <- t(summary_mn$coefficients)
    colnames(coefs) <- c('MCI', 'AD')

    wald_stats <- t(abs(summary_mn$coefficients) / summary_mn$standard.errors)
    colnames(wald_stats) <- c('MCI', 'AD')

    wald_results <- data.frame(wald_stats) %>%
        filter(MCI > 2 | AD > 2)

    num_sig_coefs <- wald_results %>%
            pivot_longer(cols=c(MCI, AD), values_to = "value") %>%
            summarise(n = sum(value > 2))

    return(list(coefs=coefs, wald_stats=wald_stats, wald_results=wald_results, num_sig_coefs=num_sig_co
}
```

```r
bart_test <- function(pca_data) {
    class_data <- split(pca_data$X_train_pca, pca_data$y_train)

    # Calculate variances of each column in each class
    variances <- matrix(NA, nrow = ncol(pca_data$X_train_pca),
                        ncol = length(class_data),
                        dimnames = list(NULL,c("CN", "MCI", "AD")))

    for (i in 1:length(class_data)) {
      vars <- sapply(class_data[[i]], var)
      variances[,i] <- vars
    }

    var_df <- data.frame(variances) %>%
                pivot_longer(everything(), names_to='Group', values_to='Var')

    bt <- bartlett.test(x=var_df$Var, g=var_df$Group)
    print(bt)

    alpha <- 0.05
```

```
    if (bt$p.value < alpha) {
        print('We REJECT the null that the variances are the same across all classes')
        } else {
        print('We FAIL TO REJECT the null that the variances are the same across all classes')
        }

    return(list(var_df=var_df, class_data=class_data))
    }



ks_test <- function(pca_data) {
    class_data <- split(pca_data$X_train_pca, pca_data$y_train)

    alpha <- 0.05
    # define a function to perform the KS test for normality on each column of a data frame
    ks_test <- function(x) {
      ks.test(x, "pnorm")$p.value
    }

    # apply the KS test to each column of each subset
    results <- lapply(class_data, function(subset) {
      lapply(subset, ks_test)
    })

    ks_df <- data.frame(cbind(results$`0`, results$`1`, results$`2`))
    colnames(ks_df) <- c('CN', 'MCI', 'AD')

    normal_pcs <- ks_df %>%
        filter(any() >= alpha)

    non_normal_pcs <- ks_df %>%
        filter(any() < alpha)

    return(list(ks_df=ks_df,normal_pcs=normal_pcs,
           non_normal_pcs=non_normal_pcs,
           non_normal_pcs=non_normal_pcs))
}
```