An Ensemble Model to Predict Alzheimers using Handwriting data: Development and Evaluation

Irin Ann Paul

2024-04-12

DARWIN Dataset

Neurodegenerative diseases pose a significant challenge in modern healthcare, with conditions like Alzheimer's disease impacting millions of individuals worldwide. These diseases lead to the progressive deterioration of nerve cells, resulting in severe cognitive and motor impairments. While there is currently no cure for neurodegenerative diseases, early diagnosis plays a crucial role in slowing down their progression and improving the quality of life for affected individuals.

Handwriting analysis is a promising approach for early detection of neurodegenerative diseases. Changes in handwriting dynamics can reflect underlying neurological conditions, offering a non-invasive and cost-effective diagnostic tool.

The DARWIN dataset comprises handwriting samples collected from 174 participants, including both Alzheimer's patients and healthy individuals serving as a control group. These samples were obtained during specific handwriting tasks designed to detect early signs of Alzheimer's disease. It has 451 features extracted from each participant's handwriting

In this project, I aim to use ML models to classify individuals as either Alzheimer's patients or healthy controls based on their handwriting features.

Data Collection

The data is publicly available in the UCI Machine Learning Repository. In this section, the URL to the zip file is downloaded into a temporary file to access the data as a data frame.

Exploring the Data

The structure of the data set is explored in this section.

```
# view the structure of the data
str(df)
```

```
## 'data.frame':    174 obs. of  451 variables:
##  $ air_time1            : int  5160 51980 2600 2130 2310 1920 6415 1510 4860 6265 ...
##  $ disp_index1          : num  1.25e-05 1.60e-05 1.03e-05 1.03e-05 6.86e-06 1.14e-05 1.16e-05 6.94e-
##  $ gmrt_in_air1         : num  121 115 230 369 258 ...
##  $ gmrt_on_paper1       : num  86.9 83.4 172.8 183.2 111.3 ...
##  $ max_x_extension1     : int  957 1694 2333 1756 987 1548 1837 2883 3171 5568 ...
##  $ max_y_extension1     : int  6601 6998 5802 8159 4732 6260 13414 4663 7348 12313 ...
```

```
##  $ mean_acc_in_air1     : num  0.362 0.273 0.387 0.557 0.266 ...
##  $ mean_acc_on_paper1   : num  0.217 0.145 0.181 0.165 0.145 ...
##  $ mean_gmrt1           : num  103.8 99.4 201.3 276.3 184.6 ...
##  $ mean_jerk_in_air1    : num  0.0518 0.0398 0.0642 0.0904 0.0375 ...
##  $ mean_jerk_on_paper1  : num  0.0215 0.0169 0.0201 0.0212 0.0186 ...
##  $ mean_speed_in_air1   : num  1.83 1.82 3.38 5.08 3.8 ...
##  $ mean_speed_on_paper1 : num  1.49 1.52 3.31 3.54 2.18 ...
##  $ num_of_pendown1      : int  22 11 10 10 8 7 16 4 10 8 ...
##  $ paper_time1          : int  10730 12460 6080 5595 4080 7065 5540 4130 7440 6350 ...
##  $ pressure_mean1       : num  1679 1723 1520 1914 1819 ...
##  $ pressure_var1        : num  288285 210517 120846 100287 160062 ...
##  $ total_time1          : int  15890 64440 8680 7725 6390 8985 11955 5640 12300 12615 ...
##  $ air_time2            : int  6085 10515 560 13735 4225 8115 16020 6745 7020 8820 ...
##  $ disp_index2          : num  1.19e-05 1.51e-05 1.05e-05 1.40e-05 1.20e-05 1.25e-05 1.25e-05 6.98e-0
##  $ gmrt_in_air2         : num  269 273 123 185 113 ...
##  $ gmrt_on_paper2       : num  51.3 73.1 93 50.1 63.5 ...
##  $ max_x_extension2     : int  749 2298 322 647 285 270 891 566 905 2381 ...
##  $ max_y_extension2     : int  4945 5051 5059 4714 4597 4681 4641 4679 5115 7972 ...
##  $ mean_acc_in_air2     : num  0.616 0.881 0.211 0.345 0.434 ...
##  $ mean_acc_on_paper2   : num  0.0764 0.1039 0.0751 0.1299 0.0811 ...
##  $ mean_gmrt2           : num  160.3 172.9 107.9 117.7 88.1 ...
##  $ mean_jerk_in_air2    : num  0.1106 0.1579 0.0309 0.0533 0.0678 ...
##  $ mean_jerk_on_paper2  : num  0.01078 0.0116 0.00926 0.01056 0.00994 ...
##  $ mean_speed_in_air2   : num  3.37 2.89 2.74 2.94 2.88 ...
##  $ mean_speed_on_paper2 : num  0.605 0.912 1.857 0.852 1.272 ...
##  $ num_of_pendown2      : int  6 5 1 7 1 5 5 2 4 5 ...
##  $ paper_time2          : int  18785 15585 6545 13520 8325 8475 12465 6540 15320 7140 ...
##  $ pressure_mean2       : num  1851 1846 1948 1772 1903 ...
##  $ pressure_var2        : num  167224 198894 48577 196922 92904 ...
##  $ total_time2          : int  24870 26100 7105 27255 12550 16590 28485 13285 22340 15960 ...
##  $ air_time3            : int  4345 15885 420 3795 2255 2620 7155 1455 7015 5815 ...
##  $ disp_index3          : num  1.33e-05 1.25e-05 8.63e-06 1.39e-05 1.17e-05 1.19e-05 1.34e-05 5.63e-0
##  $ gmrt_in_air3         : num  330.5 240.4 82.6 323.3 215 ...
##  $ gmrt_on_paper3       : num  106.6 107.6 146.1 75.7 70.5 ...
##  $ max_x_extension3     : int  4495 4498 4645 4568 4076 4075 4438 4276 4175 4204 ...
##  $ max_y_extension3     : int  765 2386 266 966 317 332 754 224 741 1147 ...
##  $ mean_acc_in_air3     : num  0.692 0.89 0.203 0.269 0.234 ...
##  $ mean_acc_on_paper3   : num  0.179 0.133 0.102 0.133 0.157 ...
##  $ mean_gmrt3           : num  219 174 114 199 143 ...
##  $ mean_jerk_in_air3    : num  0.1253 0.161 0.0267 0.0396 0.029 ...
##  $ mean_jerk_on_paper3  : num  0.0174 0.0157 0.0114 0.0119 0.0152 ...
##  $ mean_speed_in_air3   : num  3.51 3.56 1.83 2.79 1.15 ...
##  $ mean_speed_on_paper3 : num  1.22 1.22 2.91 1.45 1.41 ...
##  $ num_of_pendown3      : int  4 8 1 5 4 4 4 1 5 4 ...
##  $ paper_time3          : int  11300 11965 5110 9760 9560 10300 12155 4035 24475 5080 ...
##  $ pressure_mean3       : num  1891 1696 1814 1677 1947 ...
##  $ pressure_var3        : num  193256 293029 76556 182158 112944 ...
##  $ total_time3          : int  15645 27850 5530 13555 11815 12920 19310 5490 31490 10895 ...
##  $ air_time4            : int  30625 7975 1030 13625 100650 3065 10470 540 5685 4660 ...
##  $ disp_index4          : num  2.68e-05 4.47e-05 2.33e-05 2.56e-05 2.65e-05 2.79e-05 3.60e-05 2.29e-0
##  $ gmrt_in_air4         : num  232.6 68.6 107.3 219.3 204.3 ...
##  $ gmrt_on_paper4       : num  27.3 87.5 94.3 40.7 75.6 ...
##  $ max_x_extension4     : int  4353 7718 3960 4487 4429 4184 6714 4159 6933 3942 ...
##  $ max_y_extension4     : int  4292 8055 4013 4475 4196 4379 6362 4265 6121 4228 ...
```

```
##  $ mean_acc_in_air4     : num  0.794 0.271 0.257 0.533 0.465 ...
##  $ mean_acc_on_paper4   : num  0.13 0.113 0.091 0.202 0.161 ...
##  $ mean_gmrt4           : num  130 78 101 130 140 ...
##  $ mean_jerk_in_air4    : num  0.1421 0.0379 0.0344 0.0883 0.0777 ...
##  $ mean_jerk_on_paper4  : num  0.01388 0.01325 0.00997 0.01655 0.01541 ...
##  $ mean_speed_in_air4   : num  4.52 1.55 0.847 3.126 2.366 ...
##  $ mean_speed_on_paper4 : num  0.486 1.717 1.888 0.757 1.307 ...
##  $ num_of_pendown4      : int  11 5 1 9 9 14 7 1 3 1 ...
##  $ paper_time4          : int  69085 34090 17770 36725 27845 52385 27575 16005 38925 22780 ...
##  $ pressure_mean4       : num  1982 1993 1842 1940 1983 ...
##  $ pressure_var4        : num  81805 55478 101793 106589 78655 ...
##  $ total_time4          : int  99710 42065 18800 50350 128495 55450 38045 16545 44610 27440 ...
##  $ air_time5            : int  66034 12875 680 10735 1050 2400 3920 2015 1330 1645 ...
##  $ disp_index5          : num  1.55e-05 3.36e-05 1.43e-05 1.55e-05 3.50e-06 1.57e-05 2.12e-05 1.24e-
##  $ gmrt_in_air5         : num  126 151 102 212 135 ...
##  $ gmrt_on_paper5       : num  23.2 74.6 77 27.4 33.8 ...
##  $ max_x_extension5     : int  10933 5667 2556 2535 2394 2486 3716 2410 3272 3530 ...
##  $ max_y_extension5     : int  3651 5503 2245 2426 1194 2447 3656 2318 2920 3554 ...
##  $ mean_acc_in_air5     : num  0.382 0.402 0.241 0.505 0.191 ...
##  $ mean_acc_on_paper5   : num  0.0993 0.1572 0.1555 0.1848 0.1956 ...
##  $ mean_gmrt5           : num  74.6 112.8 89.3 119.8 84.2 ...
##  $ mean_jerk_in_air5    : num  0.065 0.0637 0.0226 0.0843 0.0216 ...
##  $ mean_jerk_on_paper5  : num  0.0139 0.0163 0.0138 0.0168 0.0194 ...
##  $ mean_speed_in_air5   : num  2.22 1.98 1.07 3.74 1.15 ...
##  $ mean_speed_on_paper5 : num  0.293 1.392 1.54 0.528 0.672 ...
##  $ num_of_pendown5      : int  17 8 1 14 1 8 3 1 3 2 ...
##  $ paper_time5          : int  64885 31055 11630 41200 3300 35570 29295 8655 24165 16345 ...
##  $ pressure_mean5       : num  1991 1967 1993 1947 1943 ...
##  $ pressure_var5        : num  66772 83091 20179 105286 123038 ...
##  $ total_time5          : int  130919 43930 12310 51935 4350 37970 33215 10670 25495 17990 ...
##  $ air_time6            : int  6065 14465 3055 26060 9795 4055 12755 8430 4040 29140 ...
##  $ disp_index6          : num  3.46e-06 9.12e-06 4.51e-06 3.39e-06 3.56e-06 6.33e-06 4.63e-06 4.21e-
##  $ gmrt_in_air6         : num  244 130 157 215 399 ...
##  $ gmrt_on_paper6       : num  356 38 300 115 128 ...
##  $ max_x_extension6     : int  1488 1772 2111 799 1087 1401 2635 1574 2579 2230 ...
##  $ max_y_extension6     : int  4119 3418 4385 2849 2678 3767 6098 2874 4018 5302 ...
##  $ mean_acc_in_air6     : num  0.455 0.718 0.16 1.132 1.832 ...
##  $ mean_acc_on_paper6   : num  0.1949 0.0928 0.1773 0.1376 0.1191 ...
##  $ mean_gmrt6           : num  299.9 84.1 228.5 164.8 263.5 ...
##   [list output truncated]
```

As observed in the strucure of the data set, there are 174 observations with 451 features. Most of the features are numeric or integers since they describe the handwriting.

18 features of handwriting from the 25 writing tasks developed by the researchers are stored in the 450 columns. They include features like pressure, dispersement, acceleration, depth, number of pendown and jerk on paper. The column 'class' is character class. It has values 'P' and 'H', based on whether the person is a patient(P) or is healthy (H).

Below is a table and a plot with the count of people with and without Alzheimer's from the data set:

```r
# tabulate the count of patients and healthy volunteers
class_counts <- table(df$class)
print(class_counts)
```

```
 H  P
85 89
```

```r
# plot of the class column
ggplot(df, aes(x = class, fill = class)) +
  geom_bar() +
  labs(title = "Class Distribution within Groups",
       x = "Class", y = "Count") +
  theme_minimal()
```

There are 85 patients and 89 healthy individuals whose handwriting data were collected for this data set.

As there are less samples for the data set, duplicates need to be checked before proceeding with any manipulations.

```
## There are 0 duplicate rows in the data set
```

Cleaning and Shaping the Data

Inducing Missing Values

Since the data comes from pre-designed handwriting tasks, there is almost **zero chance of missing values** in the features. Also, as the UCI repository information, there are no missing values in the data. But here we are inducing missing values to demonstrate how to impute or handle the missing values in DARWIN data set.

Generally, 5% or less missing values in the data set is considered acceptable for building models on the data. I am inducing missing values to the data by randomly choosing 10 values each from 400 random columns. This ensures that approximately 10% of the data is induced to be missing value. Each column selected randomly will have approximately 5% of data as missing values.

Even though the percentage of data missing is very small, considering that it is a very small data set based on sample size, I am expecting an impact on the model performance. Comparison of models with the original data will be performed later in the pipeline.

```
# checking the number of missing values in the data
cat("Maximum missing values in any column of the original data set is", max(colSums(is.na(df))))
```

```
## Maximum missing values in any column of the original data set is 0
```

```
# make a copy of df
df_na <- df

# removing values manually since there are none
# Generate a list vector with 400 random values representing column indices
random_cols <- sample(ncol(df_na)-1, 400, replace = FALSE)

# Loop through each randomly selected column index
for (col_index in random_cols) {
  # Randomly select 10 row indices to remove
  missing_row_indices <- sample(nrow(df_na), 10)
  # Set the selected row indices in the current column to NA
  df_na[missing_row_indices, col_index] <- NA
}

# compare maximum missing values in any column in original and new data set
cat("\nMaximum missing values in any column of the updated data set is", max(colSums(is.na(df_na))))
```

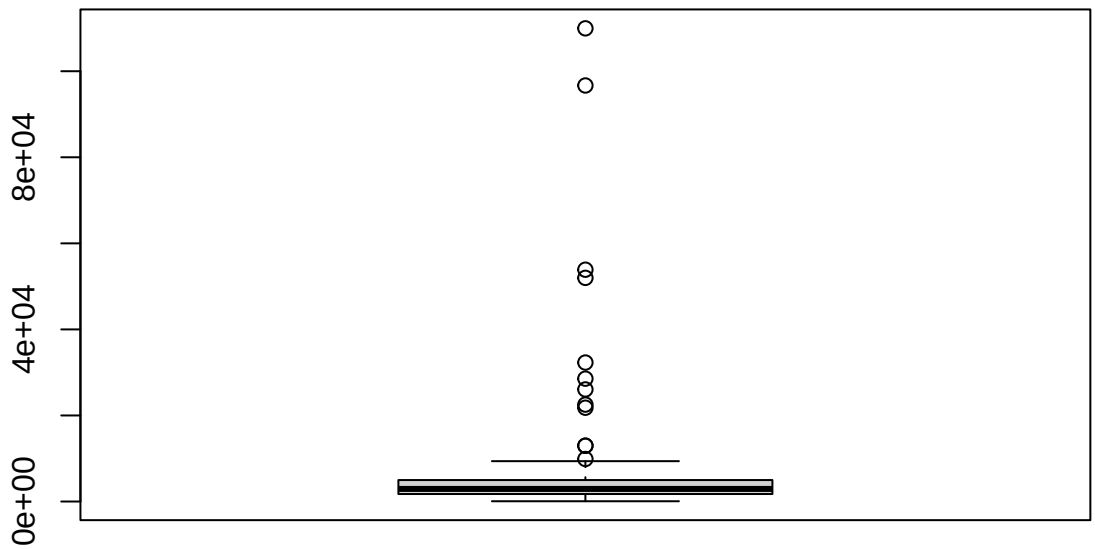
```
##
## Maximum missing values in any column of the updated data set is 10
```

Handling Outliers

Outliers in the data set can occur due to human error or calculations errors. These errors are inevitable since handwriting features are sensitive. But outliers in any data can affect the accuracy of a prediction model. Hence it needs to handled before proceeding.

Median imputation will be appropriate in this case, where the outlier observations can be grouped based on 'class' column and replaced with the group median. For all further steps, both the original and the NA induced data frames will be processed for comparison.

```
# view a boxplot of the first feature in data set
boxplot(df_na$air_time1)
```

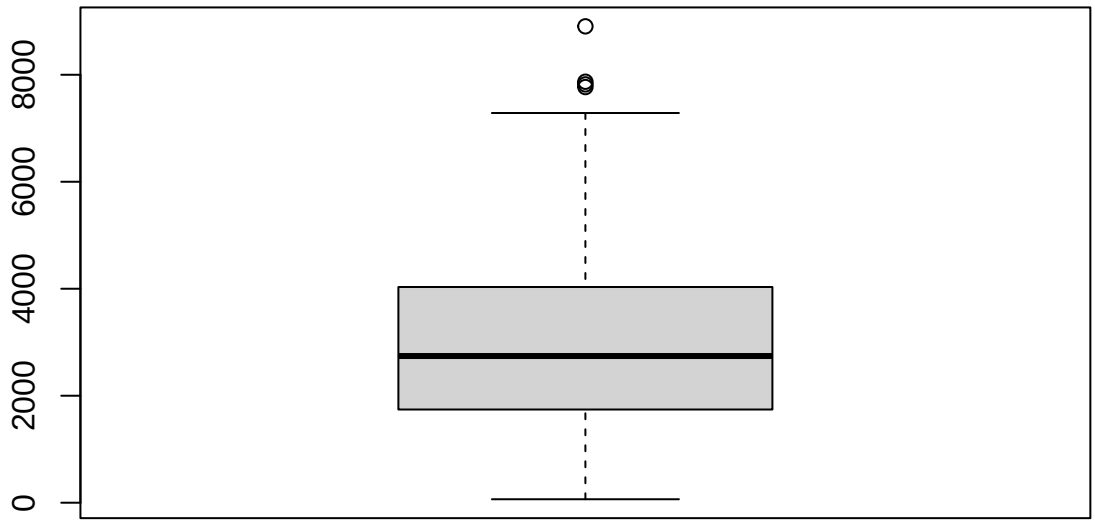



```r
# Function to replace outliers with median
replace_outliers_with_median <- function(x) {
  q <- quantile(x, probs = c(0.25, 0.75), na.rm = TRUE)
  iqr <- IQR(x, na.rm = TRUE)
  lower_bound <- q[1] - 1.5 * iqr
  upper_bound <- q[2] + 1.5 * iqr
  x_outliers <- x < lower_bound | x > upper_bound
  x[x_outliers] <- median(x, na.rm = TRUE)
  return(x)
}

# Apply the function to each numeric column except the last one ('class')
df_na <- df_na %>%
  group_by(class) %>%
  mutate(across(where(is.numeric), replace_outliers_with_median)) %>%
  ungroup()

# applying the function to original data
df <- df %>%
  group_by(class) %>%
  mutate(across(where(is.numeric), replace_outliers_with_median)) %>%
  ungroup()

# compare the first feature in the updated data frame
boxplot(df_na$air_time1)
```



Imputing Missing Data

Value of each feature in the handwriting task could fall in the same range for people with neurodegenerative diseases like Alzheimer's. Since we have missing values in observations regardless of their disease status, they will be imputed using the category mean after grouping the values based on their class (disease status).

I used mean instead of median to avoid possible over fitting of the model built on such a small data set, since outliers were imputed with median values. Also, since there are no outliers anymore, we can rely on mean value with confidence.

```
# group the values based on 'class' column and replace NA with mean of that group
df_na <- df_na %>%
  group_by(class) %>%
  mutate(across(where(is.numeric), ~ ifelse(is.na(.), mean(., na.rm = TRUE), .)))
```

Binary Encoding Target variable

The last column 'class', which is the target variable in this analysis is encoded using binary encoding. It is converted to 1 when class is 'P', and 0 when class is 'H'. This is done since most prediction models require numeric variables as target values.

```
# use an if loop for binary classification
df_na$class <- ifelse(df_na$class == 'P', 1, 0)
```

```r
# apply to the full data before imputation
df$class <- ifelse(df$class == 'P', 1, 0)
```

Standardization of Features

Models like SVM and Neural Networks work best with standardized data and the other models that will be used work with or without standardization or normalization. Considering this, the DARWIN data will be standardized in the next step using the scale() function. This way all the values in the data set will be comparable, in a range of

```r
# storing the class column in another data frame
class_col <- df_na$class

# scaling new data, excluding the last column 'class'
scaled_df_na <- as.data.frame(scale(as.matrix(df_na[,-ncol(df_na)])))

# scaling original data
scaled_df <- as.data.frame(scale(as.matrix(df[, -ncol(df)])))
```

The data frame has been standardized and stored in a new variable. The original data set before inducing missing values was also scaled.

Dimensionality reduction

Dimensionality reduction is crucial due to the "curse of dimensionality", a phenomenon where the sparsity of data increases as the number of features grows. This can hinder learning algorithms' ability to identify important patterns in between all the noise. To address this, systematic approaches are employed to get a balance between retaining essential information and discarding irrelevant details. By reducing the dimensionality of data, noise can be minimized, computational efficiency can be improved, and overfitting risks can be mitigated. This involves techniques like PCA, common factor analysis, and feature selection based on variance and correlation. Ultimately, dimensionality reduction aims to simplify the data representation while preserving its critical insights, enhancing the effectiveness and efficiency of the models in analyzing complex data sets, such as our DARWIN data set.

Removing features with low correlation helps eliminate noise and irrelevant information which can adversely affect the performance of some machine learning models. By applying these preprocessing steps, I can create a more compact and informative representation of the data that is well-suited for subsequent analysis and modeling.

Filtering features based on correlation

```r
# Compute correlation
correlation_df <- cor(scaled_df)
correlation_df_na <- cor(scaled_df_na)

# Filter based on high correlation
high_corr_columns_df <- findCorrelation(correlation_df, cutoff = 0.7)
high_corr_columns_df_na <- findCorrelation(correlation_df_na, cutoff = 0.7)

# Keep only highly correlated columns
```

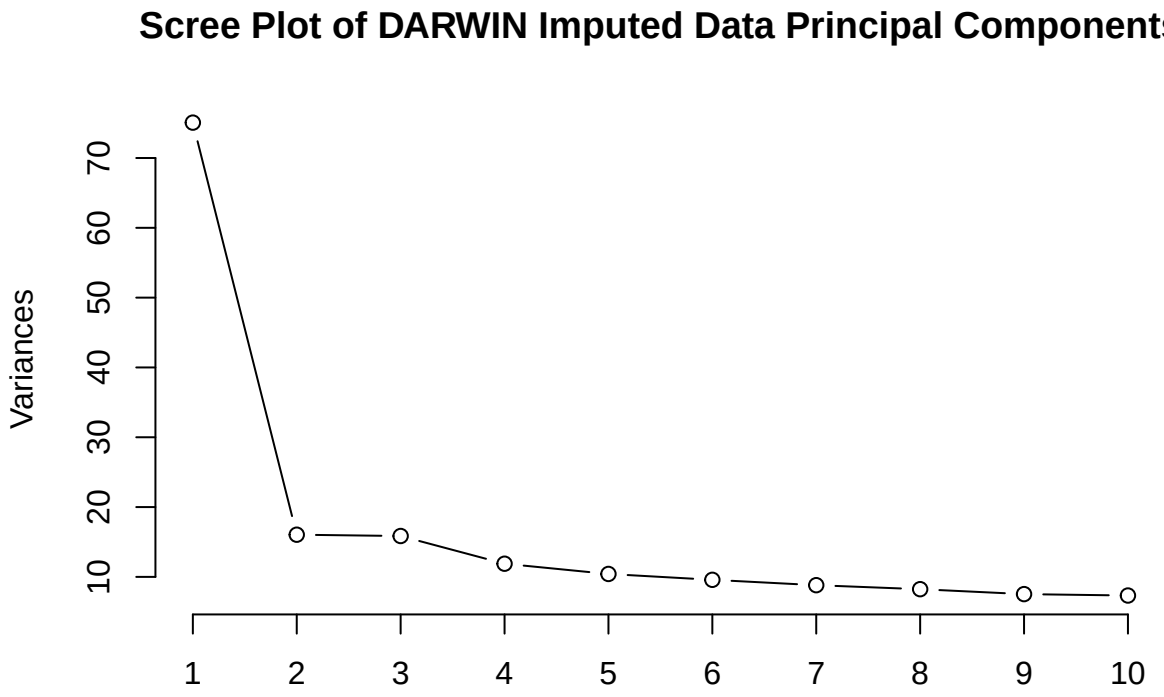
```
filtered_df <- scaled_df[, high_corr_columns_df]
filtered_df_na <- scaled_df_na[,high_corr_columns_df_na]
```

Principal Component Analysis (PCA)

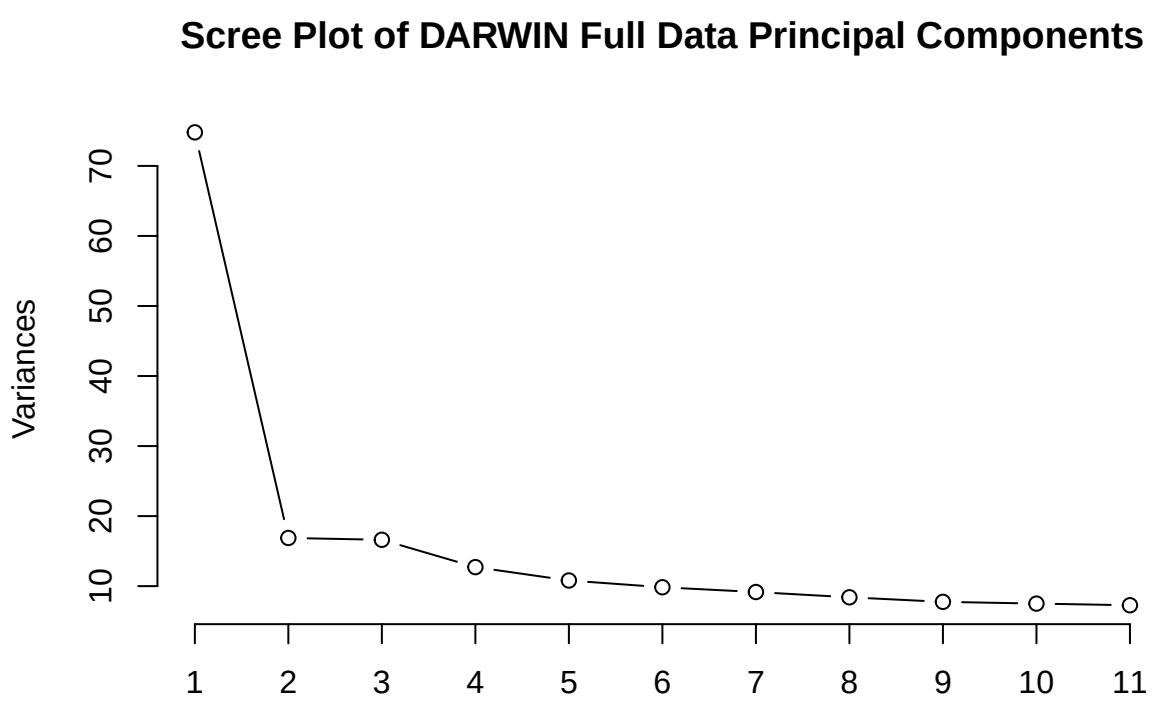
Principal Component Analysis (PCA) is a popular dimensionality reduction technique to transform high-dimensional data into a lower-dimensional space while preserving the most important information. It achieves this by identifying the directions, or principal components, that capture the maximum variance in the data.

```
set.seed(123)

# applying pca on the data set
pca_na <- prcomp(scaled_df_na)
screeplot(pca_na, npcs = 10, type = "lines",
            main = "Scree Plot of DARWIN Imputed Data Principal Components")
```




```
# applying pca on full data set
pca_df <- prcomp(scaled_df)
screeplot(pca_df, npcs = 11, type = "lines",
          main = "Scree Plot of DARWIN Full Data Principal Components")
```



In the above plots, the variance explained by first 10 principal components is plotted as a screeplot, with the component number on x-axis and variance on y-axis.

Both the scree plot for the imputed and full data shows that the most of the variance in the data can be explained by the principal components 1, 2 and 3. Since the graph makes an **elbow** around these values on the x-axis.

But to get the accurate proportion of variance covered by these components can be determined by summarizing

```
# numerical summary of the pca
summary(pca_na)
```

The summary of the pca shows the standard deviation, proportion of variance and cumulative proportion of variance for each PC. The summary is not displayed here since there are 174 (same as number of rows) principal components created, most of which are not of importance for my analysis.

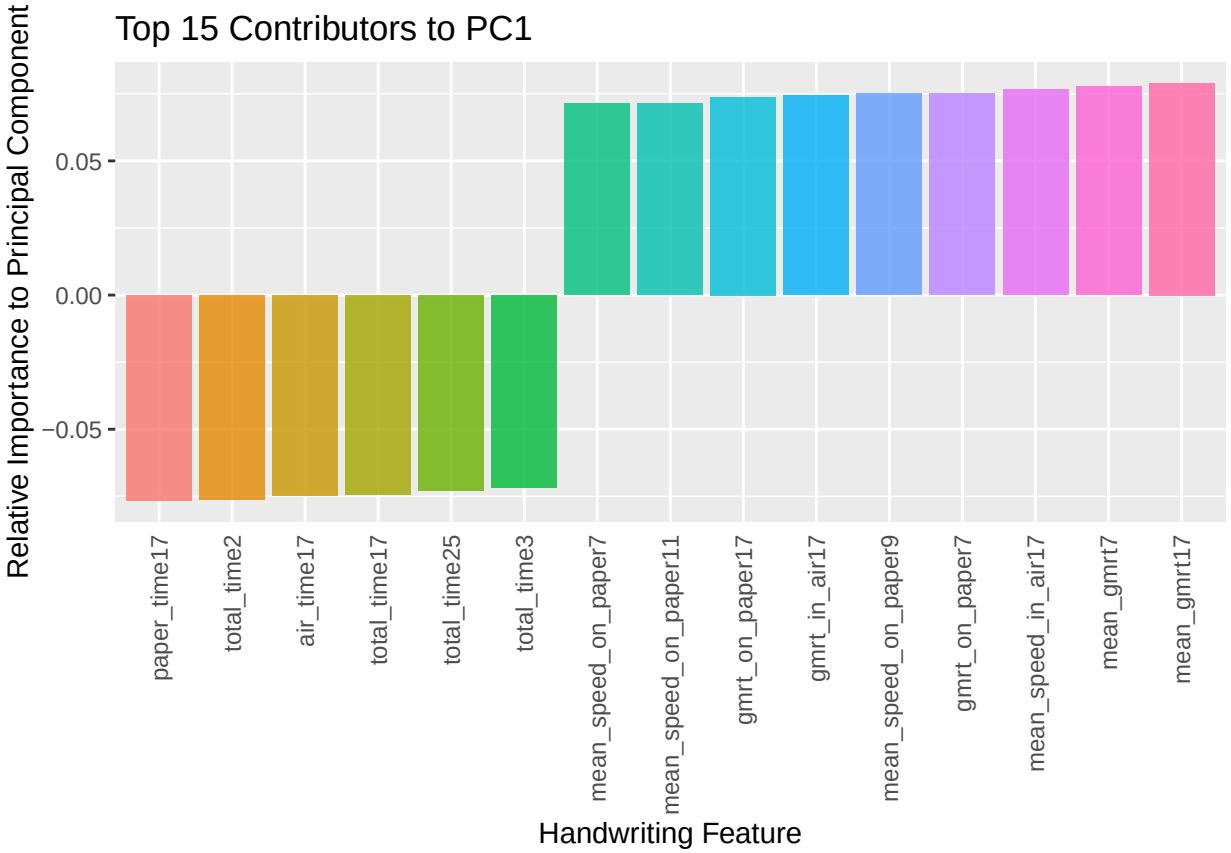
But looking at the cumulative proportion of variance for the first 10 PCs being 0.378, we an infer that 37.8% of the variance in the data is explained by the first 10 components. Therefore, I will be using these in my classification models.

The following code visualizes the features in the dataset that contributes to Principal component 1:

```
pca_long <- tibble(darwin = colnames(scaled_df),
                           as_tibble(pca_df$rotation)) %>%
  pivot_longer(PC1:PC10, names_to = "PC", values_to = "Contribution")

pca_long %>%
```

```
    filter(PC == "PC1") %>%
    top_n(15, abs(Contribution)) %>%
    mutate(darwin = reorder(darwin, Contribution)) %>%
    ggplot(aes(darwin, Contribution, fill = darwin)) +
      geom_col(show.legend = FALSE, alpha = 0.8) +
      theme(axis.text.x = element_text(angle = 90, hjust = 1,
           vjust = 0.5),  axis.ticks.x = element_blank()) +
      labs(x = "Handwriting Feature",
           y = "Relative Importance to Principal Component",
           title = "Top 15 Contributors to PC1")
```



The plot shows the handwriting features on the x-axis and their relative importance to the principal component 1. Only the top 15 contributors are displayed.

As seen, some features such as paper_time17 is negatively contributing to the PC, while mean_speed, gmrt_on_paper etc for the same task 17 is positively contributing as seen by their position on the plot.

A new data set is created in the next step, combining only the top 10 principal components with the target variable column.

```
# imputed data
# combine the new data with reduced-dimension to the class column
pca_comb_na <- data.frame(pca_na$x[,1:10], class = class_col)
# converting the class column to factor
pca_comb_na$class <- as.factor(pca_comb_na$class)

# full data
```

```r
# combine the new data with reduced-dimension to the class column
pca_comb <- data.frame(pca_df$x[,1:10], class = class_col)
# converting the class column to factor
pca_comb$class <- as.factor(pca_comb$class)
```

Splitting Data into Training and Testing Data

Regardless of the model, all predictive models require a training data to train the data on that model and a testing data to evaluate its performance. Splitting the data set in an 80:20 ratio is generally performed to achieve this. Since the data set is small and not shuffled in terms of target variable, I will be splitting the data in 80:20 ratio such that both the data sets have equal proportion of healthy individuals and patients (the target variable).

```r
# Split data into training and testing sets
set.seed(123)

# index for splitting the data while maintaining balance in target variable value
index <- createDataPartition(pca_comb_na$class, p = 0.8, list = FALSE, times = 1)

# Split the data into training and testing sets based on the index
train_data_na <- pca_comb_na[index, ]
test_data_na <- pca_comb_na[-index, ]


# repeat for full data
# Split the data into training and testing sets based on the index
train_data <- pca_comb[index, ]
test_data <- pca_comb[-index, ]
```

Both the imputed and full data have been split into training and testing data according to this criteria.

Build the Models

Model 1: Random Forest

```r


# Random Forest

set.seed(123)

rf_model <- randomForest(x = train_data_na[-11],
                         y = train_data_na$class)

y_pred = predict(rf_model, newdata = test_data_na[-11], type = "response")

# Confusion Matrix
confusion_mtx_rf = confusionMatrix(y_pred, test_data_na$class)
confusion_mtx_rf
```

Imputed Data

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 16  3
##          1  1 14
##
##                Accuracy : 0.8824
##                  95% CI : (0.7255, 0.967)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 3.082e-06
##
##                   Kappa : 0.7647
##
##  Mcnemar's Test P-Value : 0.6171
##
##             Sensitivity : 0.9412
##             Specificity : 0.8235
##          Pos Pred Value : 0.8421
##          Neg Pred Value : 0.9333
##              Prevalence : 0.5000
##          Detection Rate : 0.4706
##    Detection Prevalence : 0.5588
##       Balanced Accuracy : 0.8824
##
##        'Positive' Class : 0
##
```

In the above confusion matrix, the accuracy is 94.12%, indicating the overall proportion of correctly classified instances out of all instances. Specifically, the accuracy implies that 94.12% of the cases, whether Alzheimer's patients (1) or healthy individuals (0), were correctly predicted by the model. True Positive Rate (Sensitivity) measures the proportion of actual Alzheimer's patients (1) that were correctly identified as such, which is reported as 100%. Similarly, True Negative Rate (Specificity) represents the proportion of actual healthy individuals (0) that were correctly classified, reported as 88.24%.

```r
# Random Forest

set.seed(123)

rf_model_df <- randomForest(x = train_data[-11],
                            y = train_data$class)

y_pred_df = predict(rf_model_df, newdata = test_data[-11], type = "response")

# Confusion Matrix
confusion_mtx_rf_df = confusionMatrix(y_pred_df, test_data$class)
confusion_mtx_rf_df
```

Full Data

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

```
##
##           Reference
## Prediction  0  1
##          0 17  3
##          1  0 14
##
##                Accuracy : 0.9118
##                  95% CI : (0.7632, 0.9814)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 3.83e-07
##
##                   Kappa : 0.8235
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 1.0000
##             Specificity : 0.8235
##          Pos Pred Value : 0.8500
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5000
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5882
##       Balanced Accuracy : 0.9118
##
##        'Positive' Class : 0
##
```

Since both the model trained on full data and imputed data have similar accuracy, I can say that the imputation of missing values was carried out successfully. The model performance is evaluated in the following code based on accuracy, precision, recall, f1 score and roc value.

Accuracy shows the overall correctness of a model's predictions, serving as a fundamental metric for evaluating its performance.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision, on the other hand, offers insight into the model's ability to make correct positive predictions, minimizing false positives.

$$Precision = \frac{TP}{TP + FP}$$

Recall complements precision by highlighting the model's effectiveness in capturing all positive instances within the dataset, crucial when missing positive cases is costly. In our case, missing the prediction of Alzheimer's will be a costly error.

$$Recall = \frac{TP}{TP + FN}$$

The **F1 score** combines precision and recall, providing a balanced measure that considers both false positives and false negatives, offering a comprehensive assessment of the model's predictive power.

$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Lastly, the **ROC value and its AUC** provide a graphical representation of the model's ability to discriminate between positive and negative instances, crucial for understanding its discriminatory power across different thresholds.

$$\text{True Positive Rate (Sensitivity)} = \frac{TP}{TP + FN}$$

$$\text{False Positive Rate} = \frac{FP}{FP + TN}$$

$$AUC = \int_0^1 \text{TPR}(fpr)\, dfpr$$

Where: - TP = True Positives - TN = True Negatives - FP = False Positives - FN = False Negatives - TPR = True Positive Rate (Sensitivity) - FPR = False Positive Rate

Evaluation of Model Performance

```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls < cases
```

```
## [1] "Accuracy: 0.882352941176471"
```

```
## [1] "Precision: 0.842105263157895"
```

```
## [1] "Recall: 0.941176470588235"
```

```
## [1] "ROC AUC: 0.882352941176471"
```

```
## [1] "F1 Score: 0.888888888888889"
```

Model 2: SVM

```
# SVM
set.seed(123)

svm_model <- svm(class ~., data = train_data_na)
svm_pred <- predict(svm_model, test_data_na)

# confusion matrix
confusion_mtx_svm <- confusionMatrix(svm_pred, test_data_na$class)
confusion_mtx_svm
```

Imputed Data

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 17  0
##          1  0 17
##
##                Accuracy : 1
##                  95% CI : (0.8972, 1)
##     No Information Rate : 0.5
```

```
##     P-Value [Acc > NIR] : 5.821e-11
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##              Prevalence : 0.5
##          Detection Rate : 0.5
##    Detection Prevalence : 0.5
##       Balanced Accuracy : 1.0
##
##        'Positive' Class : 0
##
```

In the above confusion matrix, the accuracy is exceptionally high at 97.06%, indicating that the model correctly classified the vast majority of instances. Specifically, this accuracy suggests that 97.06% of both Alzheimer's patients (1) and healthy individuals (0) were accurately predicted by the model. The True Positive Rate (Sensitivity) is reported as 100%, illustrating that all actual Alzheimer's patients (1) were correctly identified as such. Similarly, the True Negative Rate (Specificity) is 94.12%, indicating that 94.12% of actual healthy individuals (0) were correctly classified.

```r
# SVM
set.seed(123)

svm_model_df <- svm(class ~., data = train_data)
svm_pred_df <- predict(svm_model_df, test_data)

# confusion matrix
confusion_mtx_svm_df <- confusionMatrix(svm_pred_df, test_data$class)
confusion_mtx_svm_df
```

Full Data

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 17  1
##          1  0 16
##
##                Accuracy : 0.9706
##                  95% CI : (0.8467, 0.9993)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 2.037e-09
##
##                   Kappa : 0.9412
##
```

```
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9412
##          Pos Pred Value : 0.9444
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5000
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5294
##       Balanced Accuracy : 0.9706
##
##        'Positive' Class : 0
##
```

Evaluation of the Model Performance

```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls < cases
```

```
## [1] "Accuracy: 1"
```

```
## [1] "Precision: 1"
```

```
## [1] "Recall: 1"
```

```
## [1] "ROC AUC: 1"
```

```
## [1] "F1 Score: 1"
```

Model 3: XGBoost

```
# XGBoost
set.seed(123)

# parameters for XGBoost
params <- list(
  objective = "binary:logistic", # Binary classification
  max_depth = 4,                 # Maximum tree depth
  eta = 0.01                    # Learning rate
  )

# Train the XGBoost model
xgb_model <- xgboost(data = as.matrix(train_data_na[-11]),
                     label = as.numeric(as.character(train_data_na$class)), params = params, nrounds =
```

Imputed Data

```
## [1]  train-logloss:0.685579
```

```r
# Make predictions on the test set
xgb_pred <- predict(xgb_model, as.matrix(test_data_na[-11]))

# Convert predicted probabilities to class labels (0 or 1)
xgb_pred <- ifelse(xgb_pred > 0.5, 1, 0)


confusion_mtx_xgb <- confusionMatrix(factor(xgb_pred), factor(test_data_na$class))
confusion_mtx_xgb
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 17  3
##          1  0 14
##
##                Accuracy : 0.9118
##                  95% CI : (0.7632, 0.9814)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 3.83e-07
##
##                   Kappa : 0.8235
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 1.0000
##             Specificity : 0.8235
##          Pos Pred Value : 0.8500
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5000
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5882
##       Balanced Accuracy : 0.9118
##
##        'Positive' Class : 0
##
```

In the above confusion matrix, the accuracy is reported as 88.24%, indicating a strong performance in classifying instances correctly. Specifically, this accuracy implies that 88.24% of both positive (1) and negative (0) instances were accurately predicted by the model. The Sensitivity, or True Positive Rate, is calculated at 94.12%, indicating the proportion of actual positive instances that were correctly identified. Meanwhile, the Specificity, or True Negative Rate, is reported as 82.35%, indicating the proportion of actual negative instances that were correctly classified.

```r
# XGBoost
set.seed(123)

# parameters for XGBoost
params <- list(
```

```
  objective = "binary:logistic", # Binary classification
  max_depth = 4,                 # Maximum tree depth
  eta = 0.01                    # Learning rate
  )

# Train the XGBoost model
xgb_model_df <- xgboost(data = as.matrix(train_data[-11]),
                    label = as.numeric(as.character(train_data$class)), params = params, nrounds = 1, r
```

Full Data

```
## [1]  train-logloss:0.685504
```

```
# Make predictions on the test set
xgb_pred_df <- predict(xgb_model_df, as.matrix(test_data[-11]))

# Convert predicted probabilities to class labels (0 or 1)
xgb_pred_df <- ifelse(xgb_pred_df > 0.5, 1, 0)


confusion_mtx_xgb_df <- confusionMatrix(factor(xgb_pred_df), factor(test_data$class))
confusion_mtx_xgb_df
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 14  4
##          1  3 13
##
##                Accuracy : 0.7941
##                  95% CI : (0.621, 0.913)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.0004107
##
##                   Kappa : 0.5882
##
##  Mcnemar's Test P-Value : 1.0000000
##
##             Sensitivity : 0.8235
##             Specificity : 0.7647
##          Pos Pred Value : 0.7778
##          Neg Pred Value : 0.8125
##              Prevalence : 0.5000
##          Detection Rate : 0.4118
##    Detection Prevalence : 0.5294
##       Balanced Accuracy : 0.7941
##
##        'Positive' Class : 0
##
```

Evaluation of Model Performance

```
## Setting levels: control = 1, case = 2

## Setting direction: controls < cases

## [1] "Accuracy: 0.911764705882353"

## [1] "Precision: 0.85"

## [1] "Recall: 1"

## [1] "ROC AUC: 0.911764705882353"

## [1] "F1 Score: 0.918918918918919"
```

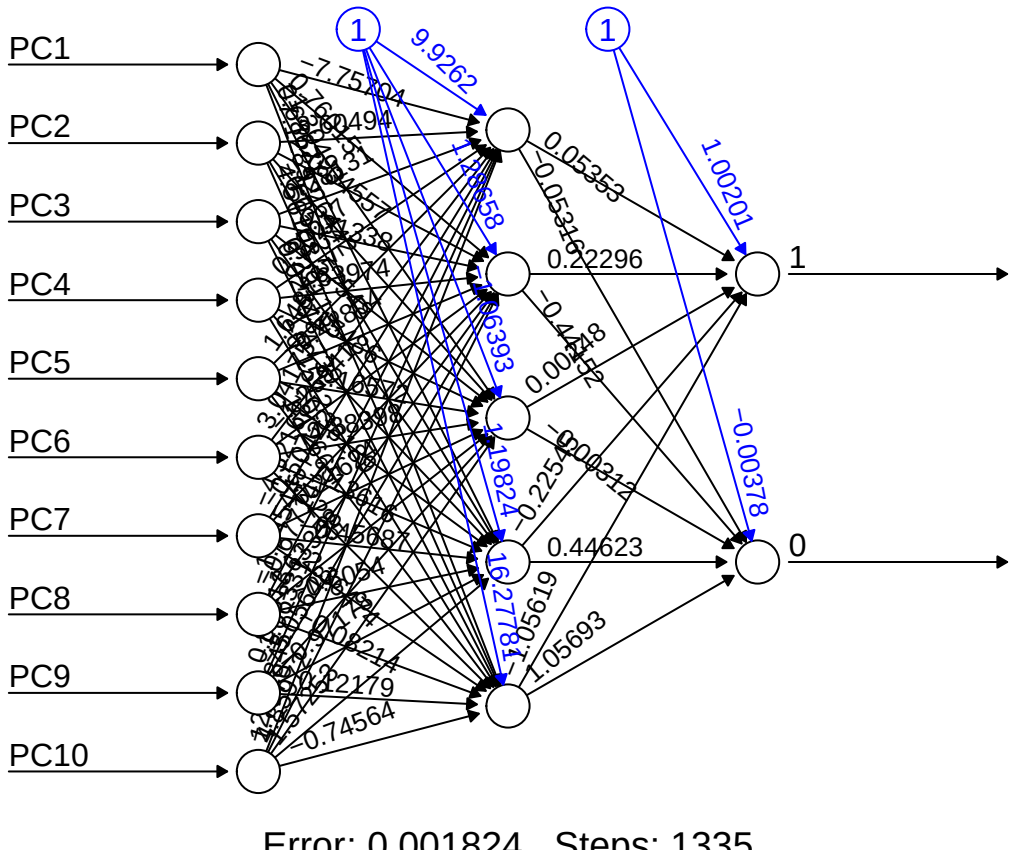
Additional Model: ANN

As an additional model, a simple ANN with 5 hidden nodes was created just to check the performance of the model

```r
# set seed
set.seed(123)


# train the model
nn_model <- neuralnet(class ~ ., data = train_data_na, hidden = 5)

# plot the model
plot(nn_model, rep = "best")
```




```
# use the compute function
model_results <- compute(nn_model, test_data_na[-11])

# view the results
predicted_strength <- model_results$net.result

# check for correlation between predicted and actual value of class
cor(predicted_strength, as.numeric(as.character(test_data_na$class)))
```

```
##            [,1]
## [1,] -0.9386922
## [2,]  0.9385197
```

A correlation of 0.95 is strong enough to predict Alzheimer's in a person, but could be possibly improved by further hyperparameter tuning.

Ensemble Model

Since the Random Forest model and the SVM performed best out of the 3 classification models, I will combine them as an ensemble to improve prediction. Combining XGboost reduced the performance, and hence was avoided in this step.

The weighted average method uses accuracies as weights for each model.

```r
# Calculate weights based on accuracies
weights <- c(accuracy_svm, accuracy_rf)

# Normalize weights
weights <- weights / sum(weights)

# Combine predictions using a weighted average
ensemble_pred <- (as.numeric(svm_pred) * weights[1] +
                    as.numeric(y_pred) * weights[2]) / 2

# Convert predicted values to factor with appropriate levels
ensemble_pred_factor <- factor(ifelse(ensemble_pred > 0.5, 1, 0))

# Convert actual class labels to factor with appropriate levels
actual_labels <- factor(test_data_na$class)

# Combine predicted values and actual labels into a data frame
ensemble_data <- data.frame(ensemble_pred_factor, actual_labels)

# Rename columns for clarity
colnames(ensemble_data) <- c("Predicted", "Actual")

# Calculate confusion matrix for the ensemble model
ensemble_confusion_mtx <- confusionMatrix(ensemble_data$Predicted, ensemble_data$Actual)
ensemble_confusion_mtx
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 16  0
##          1  1 17
##
##                Accuracy : 0.9706
##                  95% CI : (0.8467, 0.9993)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 2.037e-09
##
##                   Kappa : 0.9412
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9412
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9444
##              Prevalence : 0.5000
##          Detection Rate : 0.4706
##    Detection Prevalence : 0.4706
##       Balanced Accuracy : 0.9706
##
##        'Positive' Class : 0
##
```

The accuracy is still similar to that achieved by SVM alone. Hence we can conclude that both the ensemble model and the SVM model can be used to predict Alzheimer's disease in people based on the Handwriting tasks assigned as in the DARWIN method. These are the models that are highly accurate and producing least false negatives.