MT5758

Multivariate Analysis

University of St Andrews

Assignment 3

"Can certain typographical alterations enhance the legibility of similar characters?"

Student ID: 180005888

Group Number: 4

Introduction

The ability to easily differentiate between numbers and letters is a privilege many of us take for granted. As such, when we write we believe that all those who read our notes, papers, or exams have the same ability to read as we do. This, however, is not the case. For those with dyslexia, the act of reading is a far more daunting task. The motivation behind the project is to see if we can help improve the legibility of similar handwritten characters through typographical alterations. It is our hope that this project may act as a steppingstone for further analysis into which typographical styles and techniques should be taught to younger generations to help promote a more inclusive environment (Rello & Baeza-Yates, 2013).

To accomplish this task, we intend to examine the EMNIST dataset, a sizable collection of handwritten numbers and letters (Cohen et al., 2017). This will serve as the basis for our investigation because it is a real-world dataset that is ideal for this job due to its abundance of distinctively handwritten characters. We will use this to identify similar characters, which could be harder for readers to distinguish, and investigate if typographical alterations can improve legibility. The EMNIST dataset has high dimensionality consisting of 784 dimensions for each value, justifying the use of multivariate techniques, such as principal component analysis, to deal with this burden.

We aim to answer the research question: "Can certain typographical alterations enhance the legibility of similar characters?" Our proposed solution involves dimensionality reduction techniques (methods to reduce the complexity of high-dimensional data), clustering algorithms (grouping techniques to identify similar data points), and the implementation of typographical alterations. The impact of these alterations will be quantified using methods such as k-nearest neighbours (KNN) classification, and by calculating Euclidean distance changes between similar characters before and after the alterations.

Methods

In this section, we describe the methods used to analyse the EMNIST dataset, and attempt to answer the research question. The steps involved in the analysis are as follows:

1. Data Processing.
2. Dimensionality Reduction using PCA and NMDS.
3. Identify Potential Typographical Alterations using hierarchical clustering.
4. Propose and Implement Typographical Alterations.
5. Quantifying the Impact of Alterations using Euclidean Distance.
6. Quantifying the Impact of Alterations using KNN Classifier.

Data Processing

The EMNIST dataset consists of roughly 900,000 handwritten images that are 28x28 pixels representing digits from 0 to 9 and letters A to Z (in both upper and lower case). As the data in the EMNIST dataset contains no noise and has already been centred no pre-processing is required in relation to noise removal and centring. Nonetheless, in order to make the data easier to process, and less computationally expensive all the values will be scaled, making each pixel value of each 28 x 28 matrix lie between 0 and 1. This can be accomplished by dividing the values by 255 (the maximum pixel value).

Dimensionality Reduction via PCA and NMDS

Due to the high dimensionality of the EMNIST dataset (each image has 784 dimensions), we employed dimensionality reduction techniques to reduce the dimensionality, therefore reducing the computational burden of downstream analysis, such as clustering and classification. Dimensionality reduction techniques allow us to reduce the number of variables in a dataset, whilst retaining most of the information. There are many dimensionality reduction techniques, however, we opted to explore two, one linear method and one non-linear: Principal Component Analysis (PCA) and non-metric Multidimensional Scaling (NMDS).

PCA is a linear transformation technique, that involves projecting the original data onto a lower-dimensional space (Jolliffe & Cadima, 2016). The method generates new variables, known as Principal Components (PCs), that are uncorrelated with each other and capture the maximum variance (the most important original information). PCA performs an eigendecomposition of the data's covariance matrix, generating eigenvalues and eigenvectors. The eigenvectors are the direction of the new axes (PCs) that best capture the data's variability, whilst the corresponding eigenvalues show what amount of variance is explained by each PC. The method organises the PCs from largest to smallest eigenvalues, as such PC explains the most variance, and the second explains the second most, and so on.

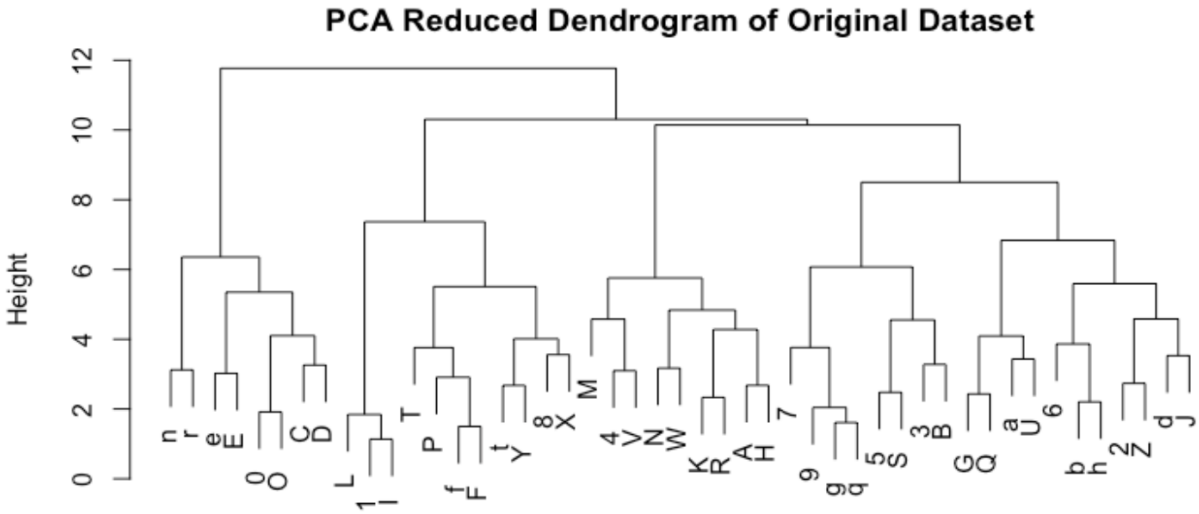
NMDS is a non-linear technique which attempts to preserve pairwise dissimilarities between data points in lower dimensions (Rabinowitz, 1975). It is an iterative approach that generates a lower-dimensional space that maintains the rank order of the original data's dissimilarity matrix pairs. The method aims to minimize a stress function that measures the discrepancy between the original pairwise dissimilarities and the distances in the reduced space. NMDS can capture complex non-linear patterns within data, which may be beneficial for the EMNIST dataset.

PCA and NMDS were both applied to the EMNIST dataset, and their performance was compared. While NMDS can potentially capture more complex relationships in the data, it is far more computationally heavy than PCA, due to the iterative nature of the process. As such, to conduct NMDS the original dataset had to be subset, this reduced the number of data points on each character, and thereby reduced the robustness of the downstream analysis. Due to the computational burden, PCA was chosen.

Identify Potential Alteration Candidates through hierarchical clustering

After reducing the data using PCA, we kept enough principal components to explain 90% of the variance in the data, which resulted in 47 PCs. This ensured that the downstream analysis results were robust whilst significantly reducing the number of dimensions from 784 to 47. We also considered selecting enough PCs to explain 95% of the variation, however, this increased the number of required PCs to 75, which was too great an increase in dimensions for the extra variance. We believe that 90% of the explained variation was optimal when trying to balance the information loss and dimensionality reduction. On a plot of explained cumulative variance, it can also be seen that at 90% we appear to be at a point of diminishing returns, otherwise known as the "elbow". The average scores for each character in the PC space served as the "centre" of the coordinates, which we used to identify similar characters and propose typographical alterations. Using these average scores, a dendrogram was constructed using hierarchical clustering (as seen in Figure 1).

Figure 1 – Dendrogram of characters



Hierarchical clustering is a technique that groups data points based on their distance from one another (Murtagh & Contreras, 2011). It is an iterative process that merges the closest pairs of data points, ultimately forming a tree-like structure called a dendrogram, where the original data points can be viewed as “leaves”. We used hierarchical clustering with the Euclidean distance metric and Ward's minimum variance method as the linkage criteria to effectively group the average scores of characters in the PC space, which represented the average location of the characters in the PC-reduced space. We chose Ward’s method because it minimizes the total within-cluster variance, resulting in more compact and well-separated clusters.

To ensure the robustness of the method of calculating the average score of each character, we required a large sample size, using PCA we were able to achieve this, having 2,400 observations per character. This sample size allowed for an approximation of a normal distribution for the data, allowing for reliable Euclidean distance calculations based on these average scores in later analyses. Calculating average scores can sometimes introduce distortions, especially when the data is not evenly distributed or contains outliers. However, the large number of independent observations per character from the EMNIST dataset allows us to expect a normal distribution for the data by the properties of the central limit theorem. This approach minimises the risk of distortions and justifies the use of the average score method.

In summary, by using PCA for dimensionality reduction and hierarchical clustering, we effectively assessed the relationships between the characters, providing a solid foundation for identifying and implementing typographical alterations to improve legibility.

Propose and Implement Typographical Alterations

After inspecting the dendrogram, we identified a series of characters that were close to one another, which may confuse those with dyslexia. For example, in the dendrogram we can see that 0 and O appear to be close to one another or Z and 2, this observation aided in identifying characters to alter. The typographical changes made were as follows:

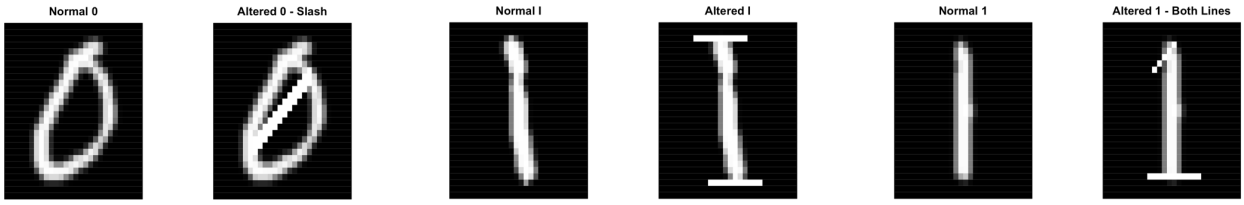
1. Adding a dot to the centre of each zero.
2. Adding a slash to each zero.
3. Adding a line through the middle of each seven.
4. Adding a horizontal line to the bottom of each one.
5. Adding an angled line to the top of each one.
6. Adding both the horizontal and angled lines to each one.
7. Adding a line through each Z.
8. Adding horizontal lines at both the top and bottom of each I.

The typographical alterations were applied to the relevant characters using several custom functions. For instance, to add a dot to the centre of each zero, we first calculated the zero's centre of mass, which is equivalent to the image's central coordinates, and then we adjusted the pixel values relative to the central coordinates to add a dot. The pixel values are simple to change because they range from 0 to 1 (due to the scaling discussed before). If we change a point's value to 1, the resulting area in the image will appear white.

With some variations, the other functions operate largely in the same way as adding a dot to each zero. For instance, when adding a line to the bottom of a one, the lowest point at which the pixel value is non-zero is located, and then the x-coordinate is obtained relative to that. This method identifies the central x-coordinate at the bottom of each one, rather than for the entire image, and applies the alteration there. This ensures that each one has a horizontal line at its bottom and in the middle.

A list of indices was created for each relevant variable, and the typographical alterations were then applied to those indexed values and saved as a new dataset to ensure the target character was altered (as seen in Figure 2).

Figure 2 – Example Characters before and after typographical alterations



Quantifying the Impact of Alterations using Euclidean Distance

To quantify the impact of the typographical alterations on legibility, we used Euclidean distance calculations. Euclidean distance is the straight-line distance between two points in an n-dimensional space, in this case, 47-dimensional space. Euclidean distance was used as a measure of similarity between the average characters. The distance measures were calculated as follows:

1. Before implementing the typographical alterations, the Euclidean distance between a character of interest (e.g., 1) and all other characters was calculated.
2. After implementing the typographical alterations, we recalculated the Euclidean distances of characters from our character of interest.
3. Using the initial Euclidean distance values from the unaltered dataset the original nearest neighbour of the character of interest was identified (being the character with the shortest distance from our character of interest), for example, 0's nearest neighbour would be O, indicating these two characters are most alike.
4. The change in Euclidean distance from its nearest neighbour after alterations were then calculated, this would indicate if the two values had become more or less similar.
5. We then created a table that displayed the percentage change in Euclidean distance for each typographical alteration. This information provided a quantitative representation of the impact of the alterations on legibility.

The use of Euclidean distance as a metric to measure improvements in legibility is justified because it quantifies character similarity (Shirkhorshidi et al., 2015). The larger the Euclidean distance between characters is, the less similar they are, which therefore means there is an improvement in legibility, as we can differentiate characters better. Conversely, a smaller Euclidean distance implies that characters are more similar and potentially harder to distinguish. By calculating the change in Euclidean distance for the characters of interest and their nearest neighbours (characters they are most likely to get confused for), we can systematically quantify the impact of the typographical alterations on character legibility.

Quantifying the Impact of Alterations using KNN Classifier

To improve upon the robustness of our quantification of whether or not the typographical changes improved legibility, we employed the use of k-nearest neighbours (KNN) classification. This is a supervised machine learning algorithm that is used to classify data points based on their similarity to one another. We opted to use this method as we can use the algorithm to calculate its accuracy in predicting character values when the characters are unaltered, and we can then see how this accuracy score changes after we implement the typographical alterations (Huque et al., 2019). The method is as follows:

1. Create a function that splits the dataset into training and testing datasets, and then trains the classifiers.
2. Use k-fold cross-validation to select the optimal k value for the algorithm.
3. Train the classifier using the original dataset.
4. Train various classifiers using the altered datasets (where typographical alterations were implemented).

5. Calculate the classification accuracy for both the altered characters and their nearest neighbours (discussed previously), providing a baseline for comparison.
6. Calculate the classification accuracy for the altered characters and their nearest neighbours, using the new KNN classifiers.
7. Compute the percentage change in classification accuracy for both the altered characters and their nearest neighbours.

The classification algorithm works by predicting what character a given value is based on how similar that point is to others like it, from the trained dataset. When generating new predictions, the algorithm aims to assign a label to the new testing data point.

The use of a KNN classification allows us to see what impact the changes have on character legibility. This acts as a second data point to our analysis, providing quantification as to whether the alterations improved legibility. We calculate an accuracy score by dividing the number of correct predictions by the total number of predictions made. If we see a positive percentage change in the classification of the characters, we will know our changes improved legibility. Furthermore, if we see a positive change for the character's nearest neighbours we can infer the same result, as that would mean our changes improved the differentiability of the characters, thereby improving legibility.

Results

Quantifying the Impact of Alterations using Euclidean Distance

The analysis conducted using the Euclidean distance clearly illustrated the impact of the typographical alterations on legibility. Table 1 shows the percentage change of the Euclidean distance between the altered characters and their nearest neighbours, showing how legibility of the characters was improved after the alterations. As stated in the methods, a greater increase in Euclidean distance means the characters are now more distinct, improving legibility.

Table 1 – Euclidean distance percentage change after alterations

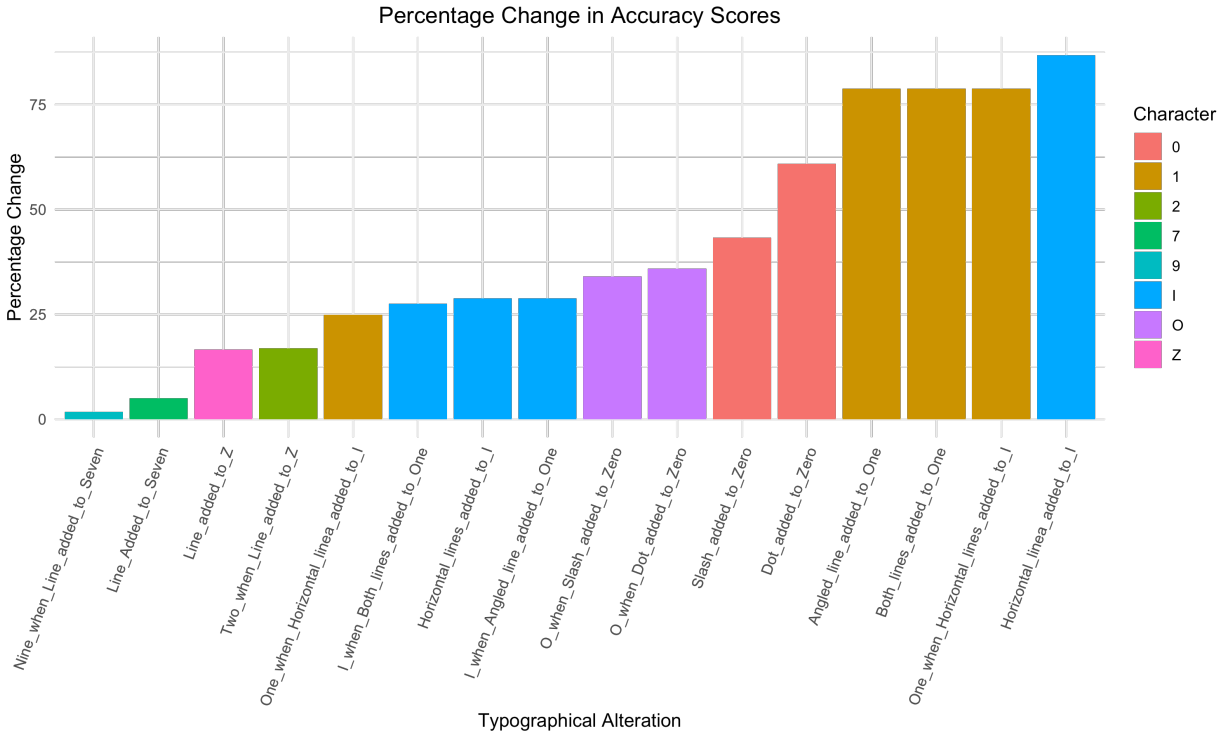
Typographical Alteration	*Nearest Neighbour*	*Percentage Change (%)*
Bottom and angled lines added to One	I	445.92
Angled line added to top of One	I	445.91
Horizontal lines added to top and bottom of I	1	441.88
Line added to bottom of One	I	440.66
Dot added to Zero	O	120.66
Line Added to Seven	9	102.56
Line added to middle of Z	2	92.18
Slash added to Zero	O	65.94

The results showed that alterations made to "I" and "1" exhibited the largest change, with the alteration of adding a horizontal line at the bottom of "1", and an angled line at the top resulting in a 445.92% increase in Euclidean distance from its nearest neighbour "I". By this metric all the changes improved legibility significantly, with the lowest change being an increase of 65.94%.

Quantifying the Impact of Alterations using KNN Classifier

The use of KNN classifiers to quantify improvements in legibility proved fruitful. As seen in Figure 3, all typographical alterations improved the classification accuracy of their own value, as well as their nearest neighbours, as measured by the accuracy score (the total correct prediction over the total number of labels).

Figure 3 – Bar chart of classification accuracy scores for respective characters



These findings further support the idea that typographical alteration, can in fact improve character legibility (as seen in Figure 3). We observe that adding a horizontal line to "I" improved the classification of its character the most, improving accuracy by 86.83%. The top and bottom lines implement on every "I" also improved the classification of "1" by 78.87%.

Discussion

The main objective of this report was to investigate if typographical alterations could help improve the legibility of certain characters, with the aim of promoting a more inclusive environment for individuals with dyslexia. The results obtained provide strong evidence that typographical alterations can improve legibility. It was shown through the Euclidean distance percentage change, that alterations to "I" and "1" had a significant impact on legibility. This result makes sense as the characters were nearly identical in the data set, both being represented as essentially straight lines. Furthermore, the improvements in the accuracy scores using the KNN classifiers further corroborate the positive effects of these typographical alterations on legibility.

The application of multivariate techniques such as PCA, and hierarchical clustering allowed us to derive meaningful insights from the high-dimensional data in a less computationally expensive way. However, these methods do have limitations and alternatives could be explored in future research. Our study, while insightful, has its limitations. For instance, we focused on a limited number of characters and alterations. Future research can explore a larger number of characters and alterations to truly see what other results are possible.

References

Cohen, G. et al. (2017) "EMNIST: Extending mnist to handwritten letters," 2017 International Joint Conference on Neural Networks (IJCNN) [Preprint]. Available at: https://doi.org/10.1109/ijcnn.2017.7966217.

Huque, A.S. et al. (2019) "Comparative study of KNN, SVM and Sr classifiers in recognizing Arabic handwritten characters employing feature fusion," Signal and Image Processing Letters, 1(2), pp. 1–10. Available at: https://doi.org/10.31763/simple.v1i2.1.

Jolliffe, I.T. and Cadima, J. (2016) "Principal component analysis: A review and recent developments," Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 374(2065), p. 20150202. Available at: https://doi.org/10.1098/rsta.2015.0202.

Murtagh, F. and Contreras, P. (2011) "Algorithms for hierarchical clustering: An overview," WIREs Data Mining and Knowledge Discovery, 2(1), pp. 86–97. Available at: https://doi.org/10.1002/widm.53.

Rabinowitz, G.B. (1975) "An introduction to nonmetric multidimensional scaling," American Journal of Political Science, 19(2), p. 343. Available at: https://doi.org/10.2307/2110441.

Rello, L. and Baeza-Yates, R. (2013) "Good fonts for dyslexia," Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility [Preprint]. Available at: https://doi.org/10.1145/2513383.2513447.

Shirkhorshidi, A.S., Aghabozorgi, S. and Wah, T.Y. (2015) "A comparison study on similarity and dissimilarity measures in clustering continuous data," PLOS ONE, 10(12). Available at: https://doi.org/10.1371/journal.pone.0144059.

Code

```
knitr::opts_chunk$set(echo = TRUE)
# Load required packages
library(tidyverse)
library(Rtsne)
library(ggthemes)
library(ggplot2)
library(class)
library(caret)
library(MASS)
library(stats)
```

```
# Load the EMNIST dataset
emnist <- read.csv("emnist-balanced-train.csv", header = FALSE)
labels <- emnist[,1]
data   <- emnist[,2:785]

# Normalise the data to 0 and 1
data <- data / 255

# Define label mapping with desired letters and their corresponding numeric labels
label_mapping <- c(0:9, LETTERS[1:26], letters[c(1:2, 4:8, 14, 17:18, 20)])
label_text <- label_mapping[labels + 1]
emnist$label_text <- label_text

# Remove duplicated rows labels
label_text <- label_text[!duplicated(data)]

# Remove duplicate rows and assign labels
data <- unique(data)
data$label_text <- label_text
```

```
# Define a function to add dot to zero
zero_dot_modifier <- function(image, radius) {

  # Convert input to a matrix
  image_matrix <- matrix(as.numeric(image), nrow = 28, ncol = 28)

  # Calculate center of mass for each axis
  rows <- 1:28
  cols <- 1:28
  x <- round(sum(colSums(image_matrix) * cols) / sum(image_matrix))
  y <- round(sum(rowSums(image_matrix) * rows) / sum(image_matrix))

  # Create a matrix with the same dimensions as image_matrix and fill it with distance values
  distance_matrix <- outer(rows, cols, function(i, j) sqrt((x - i)^2 + (y - j)^2))

  # Modify pixel values based on the distance from the center of mass
  image_matrix[distance_matrix <= radius] <- 1
  image_matrix[distance_matrix > radius & distance_matrix <= (radius + 0.5)] <- 0.5
```

```r
  return(image_matrix)
}

# Define a function to add slash to zero
zero_line_modifier <- function(image) {

  # Convert input to a matrix
  image_matrix <- matrix(as.numeric(image), nrow = 28, ncol = 28)

  # Calculate center of mass for each axis
  rows <- 1:28
  cols <- 1:28
  x <- round(sum(colSums(image_matrix) * cols) / sum(image_matrix))
  y <- round(sum(rowSums(image_matrix) * rows) / sum(image_matrix))

  # Add a slanted line through the zero using function
  slope <- -1
  y_intercept <- y - slope * x

  # Determine the start and end points of the line segment within the zero's boundary
  non_zero_indices <- which(image_matrix > 0.99, arr.ind = TRUE)
  min_col <- min(non_zero_indices[, 2])
  max_col <- max(non_zero_indices[, 2])
  start_row <- round(slope * min_col + y_intercept)
  end_row <- round(slope * max_col + y_intercept)

  # Create a matrix with the same dimensions as image_matrix and fill it with distance values
  distance_matrix <- outer(rows, cols, function(i, j) abs(i - slope * j - y_intercept))

  # Modify pixel values based on the distance from the line
  line_width <- 1
  line_shortening <- 3
  image_matrix[distance_matrix <= line_width &
                 cols >= (min_col + line_shortening) & cols <= (max_col - line_shortening) &
                 rows >= (min(start_row, end_row) + line_shortening) &
                 rows <= (max(start_row, end_row) - line_shortening)] <- 1

  return(image_matrix)
}

# Define a function to add line through seven
add_line_to_7 <- function(img) {

  # Convert input to a matrix
  image_matrix <- matrix(as.numeric(img), nrow = 28, ncol = 28)

  # Calculate center of mass for each axis
  rows <- 1:28
  cols <- 1:28
  x <- round(sum(colSums(image_matrix) * cols) / sum(image_matrix))
  y <- round(sum(rowSums(image_matrix) * rows) / sum(image_matrix))

  # Add a horizontal line through the center of the 7, ensuring it stays within bounds
```

```
  x_start <- x - 3
  x_end <- x + 10

  # While loop ensures all line coordinates stay within 28x28 dimensions
  while (x_start < 1 || x_end > 28) {
    if (x_start < 1) {
      x_start <- x_start + 1
    }
    if (x_end > 28) {
      x_end <- x_end - 1
    }
  }

  # Modify pixel values to add line
  image_matrix[x_start:x_end, y:(y + 1)] <- 1

  return(image_matrix)
}

# Define a function to line to bottom of one
add_bottom_line_to_1 <- function(img) {

  # Convert input to a matrix
  image_matrix <- matrix(as.numeric(img), nrow = 28, ncol = 28)

  # Find pixel values greater than 0.15 and use that to find bottom x
  non_zero_indices <- which(image_matrix > 0.15, arr.ind = TRUE)
  max_row <- max(non_zero_indices[, 2])

  # Calculate COM x-coordinate for max_row
  cols <- 1:28
  max_row_values <- image_matrix[, max_row]
  x_max <- round(sum(max_row_values * cols) / sum(max_row_values))

  x_start <- x_max - 5
  x_end <- x_max + 5

  # While loop ensures all line coordinates stay within 28x28 dimensions
  while (x_start < 1 || x_end > 28) {
  if (x_start < 1) {
    x_start <- x_start + 1
  }
  if (x_end > 28) {
    x_end <- x_end - 1
  }
    }

  # Modify pixel values to add line
  image_matrix[x_start:x_end, max_row] <- 1

  return(image_matrix)
}
```

```r
# Define a function to angled line to one
add_angled_line_to_1 <- function(img) {

  image_matrix <- matrix(as.numeric(img), nrow = 28, ncol = 28)

  # Find the highest non-zero row for each 1 digit
  non_zero_indices <- which(image_matrix > 0.15, arr.ind = TRUE)
  min_row <- min(non_zero_indices[, 2])
  min_col <- non_zero_indices[non_zero_indices[, 2] == min_row, 1]

  # Add a short angled line to the top of the 1, making sure it stays within bounds
  for (i in 0:4) {
    new_col <- max(min_col - i, 1)
    new_row <- min(min_row + i, 28)
    image_matrix[new_col, new_row] <- 1
  }

  return(image_matrix)
}


# Define a function to add an angled line to the top and a line to the bottom of the one
add_lines_to_1 <- function(img) {

  # Convert input to a matrix
  image_matrix <- matrix(as.numeric(img), nrow = 28, ncol = 28)

  # Find the non-zero points for lines
  non_zero_indices <- which(image_matrix > 0.15, arr.ind = TRUE)
  min_row <- min(non_zero_indices[, 2])
  min_col <- non_zero_indices[non_zero_indices[, 2] == min_row, 1]
  max_row <- max(non_zero_indices[, 2])

  # Add a short angled line to the top of the 1, making sure it stays within bounds
  for (i in 0:4) {
    new_col <- max(min_col - i, 1)
    new_row <- min(min_row + i, 28)
    image_matrix[new_col, new_row] <- 1
  }

  # Calculate COM x-coordinate for max_row
  cols <- 1:28
  max_row_values <- image_matrix[, max_row]
  x_max <- round(sum(max_row_values * cols) / sum(max_row_values))

  x_start <- x_max - 5
  x_end <- x_max + 5

  # While loop ensures all line coordinates stay within 28x28 dimensions
  while (x_start < 1 || x_end > 28) {
  if (x_start < 1) {
    x_start <- x_start + 1
  }
```

```
  if (x_end > 28) {
    x_end <- x_end - 1
  }
    }

  # Add a horizontal line to the bottom of the 1
  image_matrix[x_start:x_end, max_row] <- 1

  return(image_matrix)
}


# Define a function to add the line to the center of the z
add_line_to_z <- function(img) {

  # Convert input to a matrix
  image_matrix <- matrix(as.numeric(img), nrow = 28, ncol = 28)

  # Calculate center of mass for each axis
  rows <- 1:28
  cols <- 1:28
  x <- round(sum(colSums(image_matrix) * cols) / sum(image_matrix))
  y <- round(sum(rowSums(image_matrix) * rows) / sum(image_matrix))

  # Add a horizontal line through the center of the z, ensuring it stays within bounds
  start_x <- max(x - 9, 1)
  end_x <- min(x + 9, 28)
  image_matrix[start_x:end_x, y:(y+1)] <- 1

  return(image_matrix)
}

# Define a function to add the lines to top and bottom of I
add_line_to_I <- function(img) {

  # Convert input to a matrix
  image_matrix <- matrix(as.numeric(img), nrow = 28, ncol = 28)

  # Find the lowest and highest row with greater than 0.15 pixel value
  non_zero_indices <- which(image_matrix > 0.15, arr.ind = TRUE)
  max_row <- max(non_zero_indices[, 2])
  min_row <- min(non_zero_indices[, 2])

  # Calculate COM x-coordinate for max_row
  cols <- 1:28
  max_row_values <- image_matrix[, max_row]
  x_max <- round(sum(max_row_values * cols) / sum(max_row_values))

  # Calculate COM x-coordinate for min_row
  min_row_values <- image_matrix[, min_row]
  x_min <- round(sum(min_row_values * cols) / sum(min_row_values))
```

```
  x_start_max <- x_max - 5
  x_end_max <- x_max + 5
  x_start_min <- x_min - 5
  x_end_min <- x_min + 5

  # While loop makes sure lines are in boundary
  while (x_start_max < 1 || x_end_max > 28) {
    if (x_start_max < 1) {
      x_start_max <- x_start_max + 1
    }
    if (x_end_max > 28) {
      x_end_max <- x_end_max - 1
    }
  }


  while (x_start_min < 1 || x_end_min > 28) {
    if (x_start_min < 1) {
      x_start_min <- x_start_min + 1
    }
    if (x_end_min > 28) {
      x_end_min <- x_end_min - 1
    }
  }

  # Add a horizontal line to the top and bottom of the I
  image_matrix[x_start_max:x_end_max, max_row] <- 1
  image_matrix[x_start_min:x_end_min, min_row] <- 1

  return(image_matrix)
}
```

```
# Get indices of all variables of interest
zero_indices <- which(data$label_text == 0)
seven_indices <- which(data$label_text == 7)
one_indices <- which(data$label_text == 1)
z_indices <- which(data$label_text == "Z")
I_indices <- which(data$label_text == "I")

# Define function that generates new data set with specified modifications
modify_and_update_data <- function(indices, img_modifier, radius = NULL) {

  # Assign new data frame
  altered_data <- data[, -785]

  # Loop through specified indices and alter images
  for (idx in indices) {

    img <- matrix(data[idx, -785], nrow = 28, ncol = 28, byrow = TRUE)

    if (!is.null(radius)) {

      # When adding dot to zero include radius argument
```

```
      img_with_line <- img_modifier(img, radius)
    } else {

      # Otherwise, alter image
      img_with_line <- img_modifier(img)
    }

    # Save altered images to data frame
    altered_data[idx,] <- as.matrix(img_with_line)
  }
  return(altered_data)
}

# Generate altered datasets
altered_data <- modify_and_update_data(zero_indices, zero_dot_modifier, radius = 2.5)
altered_data2 <- modify_and_update_data(zero_indices, zero_line_modifier)
altered_data3 <- modify_and_update_data(seven_indices, add_line_to_7)
altered_data4 <- modify_and_update_data(one_indices, add_bottom_line_to_1)
altered_data5 <- modify_and_update_data(one_indices, add_angled_line_to_1)
altered_data6 <- modify_and_update_data(one_indices, add_lines_to_1)
altered_data7 <- modify_and_update_data(z_indices, add_line_to_z)
altered_data8 <- modify_and_update_data(I_indices, add_line_to_I)
```

```
# Define function to plot image
plot_image <- function(img, label) {
  img <- as.matrix(img, nrow = 28, ncol = 28, byrow = TRUE)
  dim(img) <- c(28, 28)
  img <- t(apply(img, 2, rev)) # Rotate values twice
  title <- paste("Normal", label)
  image(img, col = gray((0:255) / 255), xaxt = 'n', yaxt = 'n', main = title)
}

# Define function to plot altered image
plot_altered_image <- function(img, label) {
  img <- as.matrix(img, nrow = 28, ncol = 28, byrow = TRUE)
  dim(img) <- c(28, 28)
  img <- t(apply(img, 1, rev)) # Rotate values once
  title <- paste("Altered", label)
  image(img, col = gray((0:255) / 255), xaxt = 'n', yaxt = 'n', main = title)
}

# Plot images
par(mfrow = c(1, 2))
plot_image(data[zero_indices[1], -785], "0")
plot_altered_image(altered_data[zero_indices[1],], "0")

plot_image(data[zero_indices[1], -785], "0")
plot_altered_image(altered_data2[zero_indices[1],], "0 - Slash")

plot_image(data[seven_indices[1], -785], "7")
plot_altered_image(altered_data3[seven_indices[1],], "7")

plot_image(data[one_indices[2], -785], "1")
```

```r
plot_altered_image(altered_data4[one_indices[2],], "1")

plot_image(data[one_indices[2], -785], "1")
plot_altered_image(altered_data5[one_indices[2],], "1")

plot_image(data[one_indices[2], -785], "1")
plot_altered_image(altered_data6[one_indices[2],], "1 - Both Lines")

plot_image(data[z_indices[2], -785], "Z")
plot_altered_image(altered_data7[z_indices[2],], "Z")

plot_image(data[I_indices[2], -785], "I")
plot_altered_image(altered_data8[I_indices[2],], "I")
```

```r
# Set the sample size per label
sample_size_per_label <- 100

# Get unique labels
unique_labels <- unique(label_text)
balanced_sample <- data.frame()

# Generate small balanced subset
for (label in unique_labels) {
  label_data <- data[label_text == label, ]
  sample_indices <- sample(1:nrow(label_data), min(sample_size_per_label, nrow(label_data)))
  label_sample <- label_data[sample_indices, ]
  label_sample$label <- label_text[label_text == label][sample_indices]
  balanced_sample <- rbind(balanced_sample, label_sample)
}

# Compute the dissimilarity matrix
dissimilarity_matrix <- dist(balanced_sample[, -ncol(balanced_sample)])

# Set for reproducibility
set.seed(123)

# Perform nMDS
nmds_result <- isoMDS(dissimilarity_matrix, maxit = 1000)

# Convert the nMDS result to a data frame
nmds_df <- as.data.frame(nmds_result$points)

# Name the nMDS dimensions
names(nmds_df) <- c("nMDS1", "nMDS2")

# Add labels to the nMDS data frame
nmds_df$label <- balanced_sample$label

# Plot the nMDS result
ggplot(nmds_df, aes(x = nMDS1, y = nMDS2, color = label)) +
  geom_point() +
  theme_minimal() +
  ggtitle("Nonmetric MDS Reduction of EMNIST Dataset") +
```

```
  xlab("nMDS Dimension 1") +
  ylab("nMDS Dimension 2")
```

```
# Define function to perform PCA
perform_pca <- function(data) {

  # Remove values with near zero variance
  near_zero <- nearZeroVar(data, saveMetrics = TRUE)
  data_new <- data[, !near_zero$nzv]

  # Perform PCA
  pca <- prcomp(data_new)

  return(pca)
}

# Perform PCA
pca <- perform_pca(data[, -785])
pca2 <- perform_pca(altered_data)
pca3 <- perform_pca(altered_data2)
pca4 <- perform_pca(altered_data3)
pca5 <- perform_pca(altered_data4)
pca6 <- perform_pca(altered_data5)
pca7 <- perform_pca(altered_data6)
pca8 <- perform_pca(altered_data7)
pca9 <- perform_pca(altered_data8)
```

```
# Define function that calculates "centre" of each respective label
compute_pca_centroids <- function(pca) {

  # Compute average score from first 47 principal components
  pca_centroids <- aggregate(pca$x[, 1:47], by = list(labels = label_text), FUN = mean)

  return(pca_centroids)
}

# Perform average score calculations
pca_centroids <- compute_pca_centroids(pca)
pca_centroids2 <- compute_pca_centroids(pca2)
pca_centroids3 <- compute_pca_centroids(pca3)
pca_centroids4 <- compute_pca_centroids(pca4)
pca_centroids5 <- compute_pca_centroids(pca5)
pca_centroids6 <- compute_pca_centroids(pca6)
pca_centroids7 <- compute_pca_centroids(pca7)
pca_centroids8 <- compute_pca_centroids(pca8)
pca_centroids9 <- compute_pca_centroids(pca9)
```

```
# Define function to generate PCA dataframe
extract_pca_df <- function(pca) {
  pca_df <- as.data.frame(pca$x[, 1:47])
  pca_df$label_text <- label_text
  return(pca_df)
}
```

```r
pca_df <- extract_pca_df(pca)
pca_df2 <- extract_pca_df(pca2)
pca_df3 <- extract_pca_df(pca3)
pca_df4 <- extract_pca_df(pca4)
pca_df5 <- extract_pca_df(pca5)
pca_df6 <- extract_pca_df(pca6)
pca_df7 <- extract_pca_df(pca7)
pca_df8 <- extract_pca_df(pca8)
pca_df9 <- extract_pca_df(pca9)
```

```r
# Calculate explained variance
explained_var <- pca$sdev^2 / sum(pca$sdev^2)

# Calculate cumulative explained variance
cumulative_explained_var <- cumsum(explained_var)

# Create a data frame
explained_variance_df <- data.frame(
  PC = 1:length(explained_var),
  cum_var = cumulative_explained_var
)

# Find PC value that explains at least 90% of the variance
threshold <- 0.9
n_components <- which(cumulative_explained_var >= threshold)[1]

# Plot cumulative explained variance
ggplot(explained_variance_df, aes(x = PC, y = cum_var)) +
  geom_point() +
  geom_line() +
  geom_hline(yintercept = threshold, linetype = "dashed", color = "red") +
  geom_vline(xintercept = n_components, linetype = "dashed", color = "red") +
  labs(title = "Cumulative Explained Variance",
       x = "Principal Component",
       y = "Cumulative Explained Variance (%)") +
  ggthemes::theme_few() +
  theme(plot.title = element_text(hjust = 0.5)) +
  annotate("text", x = n_components + 25, y = threshold - 0.05,
           label = paste("PC ", n_components, "(", round(threshold * 100), "%)", sep = ""))
```

```r
# Define function to plot dendrograms
plot_dendrogram <- function(centroids, title = NULL) {

  # Set seed for reproducibility
  set.seed(123)

  # Perform hierarchical clustering
  hc <- hclust(dist(centroids[, -1]), method = "ward.D2")

  # Plot dendrogram
  plot(hc, labels = as.character(centroids$labels), main = title)
}
```

```r
plot_dendrogram(pca_centroids, "PCA Reduced Dendrogram of Original Dataset")
```

```r
# Define function to calculate and sort Euclidean distance of character of interest
sorted_distances_by_character <- function(pca_centroids, char_of_interest) {

  # Extract the coordinates of the character of interest
  char_centroid <- pca_centroids[pca_centroids$labels == char_of_interest, -1]

  # Calculate the Euclidean distance between the mean score of the
  # character of interest and all other values
  euclidean_distances <- as.matrix(dist(rbind(char_centroid, pca_centroids[, -1])))[1, -1]

  # Create a data frame
  distances_with_labels <- data.frame(labels = pca_centroids$labels,
                                      distance = euclidean_distances)

  # Sort the data frame by distance from character of interest
  sorted_distances <- distances_with_labels[order(distances_with_labels$distance),]

  return(sorted_distances)
}

# Calculate base distances, and exmaine neighbours
sorted_distances_0 <- sorted_distances_by_character(pca_centroids, "0")
sorted_distances_7 <- sorted_distances_by_character(pca_centroids, "7")
sorted_distances_Z <- sorted_distances_by_character(pca_centroids, "Z")
sorted_distances_1 <- sorted_distances_by_character(pca_centroids, "1")
sorted_distances_I <- sorted_distances_by_character(pca_centroids, "I")
```

```r
# Calculate distances with alterations
sorted_distances_0_dot <- sorted_distances_by_character(pca_centroids2, "0")
sorted_distances_0_slash <- sorted_distances_by_character(pca_centroids3, "0")

# Get required distances
O_distance <- sorted_distances_0[sorted_distances_0$labels == "O",]$distance
O_distance_dot <- sorted_distances_0_dot[sorted_distances_0_dot$labels == "O",]$distance
O_distance_slash <- sorted_distances_0_slash[sorted_distances_0_slash$labels == "O",]$distance

# Calculate change
change_dot_O <- O_distance_dot - O_distance
change_slash_O <- O_distance_slash - O_distance

# Calculate percentage change
percentage_change_dot_O <- (change_dot_O / O_distance) * 100
percentage_change_slash_O <- (change_slash_O / O_distance) * 100
```

```r
# Calculate distances with alterations
sorted_distances_7_middle_line <- sorted_distances_by_character(pca_centroids4, "7")

# Get required distances
nine_distance <- sorted_distances_7[sorted_distances_7$labels == "9",]$distance
nine_distance_middle_line <- sorted_distances_7_middle_line[
  sorted_distances_7_middle_line$labels == "9",]$distance
```

```r
# Calculate change
change_middle_line_9 <- nine_distance_middle_line - nine_distance

# Calculate percentage change
percentage_change_middle_line_9 <- (change_middle_line_9 / nine_distance) * 100
```

```r
# Calculate distances with alterations
sorted_distances_1_bottom_line <- sorted_distances_by_character(pca_centroids5, "1")
sorted_distances_1_angled_line <- sorted_distances_by_character(pca_centroids6, "1")
sorted_distances_1_both_lines <- sorted_distances_by_character(pca_centroids7, "1")

# Get required distances
I_distance <- sorted_distances_1[sorted_distances_1$labels == "I",]$distance
I_distance_bottom_line <- sorted_distances_1_bottom_line[
  sorted_distances_1_bottom_line$labels == "I",]$distance
I_distance_angled_line <- sorted_distances_1_angled_line[
  sorted_distances_1_angled_line$labels == "I",]$distance
I_distance_both_lines <- sorted_distances_1_both_lines[
  sorted_distances_1_both_lines$labels == "I",]$distance

# Calculate change
change_bottom_line_I <- I_distance_bottom_line - I_distance
change_angled_line_I <- I_distance_angled_line - I_distance
change_both_lines_I <- I_distance_both_lines - I_distance

# Calculate percentage change
percentage_change_bottom_line_I <- (change_bottom_line_I / I_distance) * 100
percentage_change_angled_line_I <- (change_angled_line_I / I_distance) * 100
percentage_change_both_lines_I <- (change_both_lines_I / I_distance) * 100
```

```r
# Calculate distances with alterations
sorted_distances_Z_middle_line <- sorted_distances_by_character(pca_centroids8, "Z")

# Get required distances
two_distance <- sorted_distances_Z[sorted_distances_Z$labels == "2",]$distance
two_distance_middle_line <- sorted_distances_Z_middle_line[
  sorted_distances_Z_middle_line$labels == "2",]$distance

# Calculate change
change_middle_line_2 <- two_distance_middle_line - two_distance

# Calculate percentage change
percentage_change_middle_line_2 <- (change_middle_line_2 / two_distance) * 100
```

```r
# Calculate distances with alterations
sorted_distances_I_lines <- sorted_distances_by_character(pca_centroids9, "I")

# Get required distances
one_distance <- sorted_distances_I[sorted_distances_I$labels == "1",]$distance
one_distance_lines <- sorted_distances_I_lines[
  sorted_distances_I_lines$labels == "1",]$distance
```

```r
# Calculate change
change_lines_1 <- one_distance_lines - one_distance

# Calculate percentage change
percentage_change_lines_1 <- (change_lines_1 / one_distance) * 100
```

```r
# Create a data frame with percentage change results
percentage_changes <- data.frame(
  Typographical_Alteration = c("Dot added to Zero", "Slash added to Zero",
                               "Line Added to Seven", "Line added to bottom of One",
                               "Angled line added to top of One",
                               "Both bottom and top lines added to One",
                               "Horizontal line added to top and bottom of I",
                               "Line added to middle of Z"),
  Character = c("O", "O", "9", "I", "I", "I", "1", "2"),
  Percentage_Change = c(percentage_change_dot_O, percentage_change_slash_O,
                        percentage_change_middle_line_9, percentage_change_bottom_line_I,
                        percentage_change_angled_line_I, percentage_change_both_lines_I,
                        percentage_change_lines_1, percentage_change_middle_line_2)
)

# Sort the data frame by percentage change (highest to lowest)
sorted_percentage_changes <- percentage_changes[order(-percentage_changes$Percentage_Change),]

# Print the sorted data frame
print(sorted_percentage_changes)

# Export the sorted data frame to a CSV file
write.csv(sorted_percentage_changes, file = "sorted_data.csv", row.names = FALSE)
```

```r
# Set for reproducibility
set.seed(123)

# Use k-folds CV to find optimal k
trControl <- trainControl(method = "cv", number = 5)

find_k <- train(label_text ~ .,
             method = "knn",
             tuneGrid = expand.grid(k = 1:20),
             trControl = trControl,
             metric = "Accuracy",
             data = pca_df)
```

```r
split_and_train_knn <- function(data_with_labels, split_ratio = 0.8, k = 5) {
  # Split the dataset into training and testing sets
  set.seed(123) # Set for reproducibility
  train_indices <- sample(1:nrow(data_with_labels), split_ratio * nrow(data_with_labels))
  train_data <- data_with_labels[train_indices, ]
  test_data <- data_with_labels[-train_indices, ]

  # Separate labels from data
  train_labels <- train_data[, ncol(train_data)]
  train_features <- train_data[, -ncol(train_data)]
```

```
  test_labels <- test_data[, ncol(test_data)]
  test_features <- test_data[, -ncol(test_data)]

  # Train the KNN classifier
  predicted_labels <- knn(train_features, test_features, cl = train_labels, k = k)

  # Return the predicted_labels and test_data
  return(list(predicted_labels = predicted_labels,
              test_data = test_data,
              test_labels = test_labels))
}

# Define function that calculates character of interest accuracy scores
calculate_accuracy_score <- function(knn_result, char_of_interest) {

  # Extract true labels
  true_labels <- knn_result$test_data$label_text

  # Calculate correct predictions for the character of interest
  correct_predictions <- sum(knn_result$predicted_labels == char_of_interest &
                             true_labels == char_of_interest)

  # Calculate the total number of labels for the character of interest
  total_labels <- sum(true_labels == char_of_interest)

  # Calculate the accuracy score
  accuracy_score <- (correct_predictions / total_labels) * 100

  return(accuracy_score)
}
```

```
# Generate KNN classifiers
knn1 <- split_and_train_knn(pca_df)
knn2 <- split_and_train_knn(pca_df2)
knn3 <- split_and_train_knn(pca_df3)
knn4 <- split_and_train_knn(pca_df4)
knn5 <- split_and_train_knn(pca_df5)
knn6 <- split_and_train_knn(pca_df6)
knn7 <- split_and_train_knn(pca_df7)
knn8 <- split_and_train_knn(pca_df8)
knn9 <- split_and_train_knn(pca_df9)
```

```
# Create set of values need for accuracy scores
classifiers_and_labels <- list(
  Zero_normal = list(knn = knn1, label = "0"),
  O_normal = list(knn = knn1, label = "O"),
  Dot_added_to_Zero = list(knn = knn2, label = "0"),
  O_when_Dot_added_to_Zero = list(knn = knn2, label = "O"),
  Slash_added_to_Zero = list(knn = knn3, label = "0"),
  O_when_Slash_added_to_Zero = list(knn = knn3, label = "O"),
  Seven_normal = list(knn = knn1, label = "7"),
  Line_Added_to_Seven = list(knn = knn4, label = "7"),
  Nine_normal = list(knn = knn1, label = "9"),
```

```
  Nine_when_Line_added_to_Seven = list(knn = knn4, label = "9"),
  One_normal = list(knn = knn1, label = "1"),
  I_normal = list(knn = knn1, label = "I"),
  Both_lines_added_to_One = list(knn = knn5, label = "1"),
  I_when_Both_lines_added_to_One = list(knn = knn5, label = "I"),
  Angled_line_added_to_One = list(knn = knn6, label = "1"),
  I_when_Angled_line_added_to_One = list(knn = knn6, label = "I"),
  One_when_Horizontal_lines_added_to_I = list(knn = knn7, label = "1"),
  Horizontal_lines_added_to_I = list(knn = knn7, label = "I"),
  Z_normal = list(knn = knn1, label = "Z"),
  Line_added_to_Z = list(knn = knn8, label = "Z"),
  Two_normal = list(knn = knn1, label = "2"),
  Two_when_Line_added_to_Z = list(knn = knn8, label = "2"),
  Horizontal_lines_added_to_I = list(knn = knn9, label = "I"),
  One_when_Horizontal_linea_added_to_I = list(knn = knn9, label = "1")
)
```

```
# Define function to calculate accuracy scores
accuracy_scores_table <- function(calculate_accuracy_score, classifiers_and_labels) {
  scores <- sapply(classifiers_and_labels, function(x) {
    calculate_accuracy_score(x$knn, x$label)
  })

  labels <- sapply(classifiers_and_labels, function(x) {
    x$label
  })

  df <- data.frame(Classifier = names(scores), Accuracy = scores, Label = labels)

  # Calculate the base accuracy scores
  base_accuracy <- tapply(df$Accuracy, df$Label, function(x) x[1])

  # Calculate the percentage change for each accuracy score relative to its base
  df$PercentageChange <- sapply(1:nrow(df), function(i) {
    (df$Accuracy[i] - base_accuracy[df$Label[i]]) / base_accuracy[df$Label[i]] * 100
  })

  sorted_df <- df[order(df$Accuracy),]

  return(sorted_df)
}

# Calculate accuracy df
accuracy_scores_df <- accuracy_scores_table(calculate_accuracy_score, classifiers_and_labels)

# Remove rows with 0 percentage change, so base values
accuracy_scores_df <- accuracy_scores_df[accuracy_scores_df$PercentageChange != 0,]

# Create a bar chart
accuracy_plot <- ggplot(ordered_accuracy_scores_df, aes(x = reorder(Classifier, PercentageChange), y = P
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Percentage Change in Accuracy Scores",
       x = "Typographical Alteration",
```

```
       y = "Percentage Change",
       fill = "Character") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 70, hjust = 1),
        plot.title = element_text(hjust = 0.5))

# Save bar chart
ggsave("accuracy_scores.png", plot = accuracy_plot, width = 10, height = 6, dpi = 300)
```