Digit Recognition using MNIST Dataset

Shriram Rajasekar, Alex Miller, Kenan Stredic

May 10, 2024

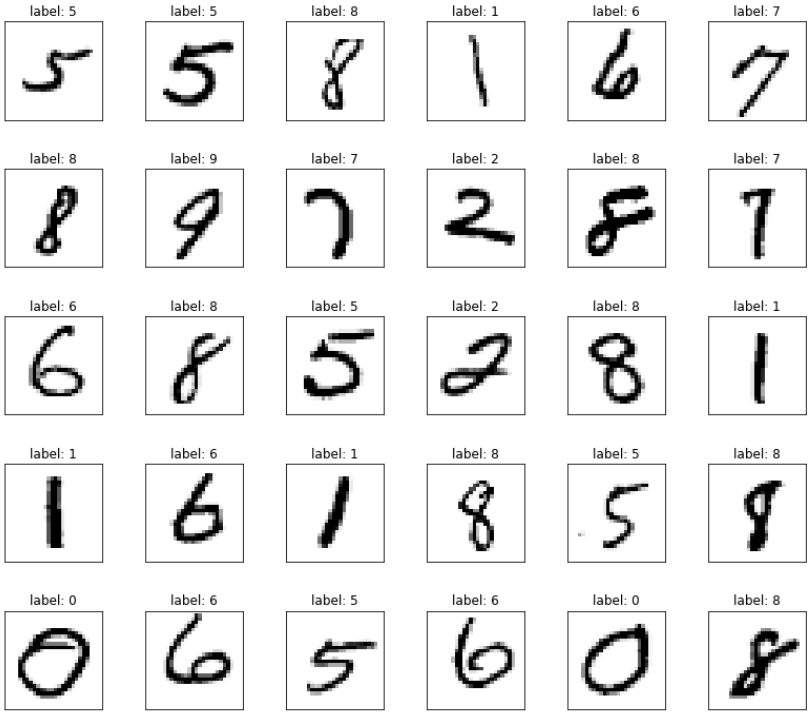
Abstract

The digit recognition task using the MNIST dataset serves as a foundational problem in the field of computer vision and machine learning. This project explores various methodologies, including semi-supervised learning techniques and clustering algorithms, to improve digit recognition accuracy. Results indicate that while semi-supervised learning approaches show promise, robust classification algorithms such as Random Forest outperform clustering methods in digit recognition tasks.

1. Introduction

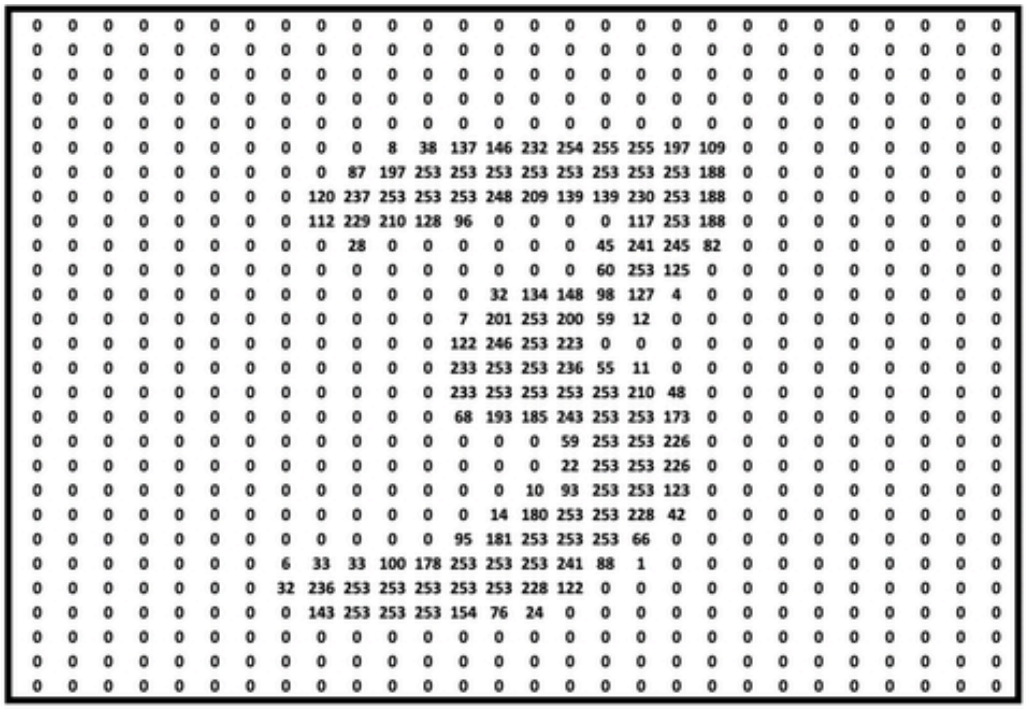
The accurate recognition of handwritten digits is a fundamental problem in machine learning, with applications ranging from optical character recognition to automated postal sorting. The MNIST dataset, comprising 70,000 handwritten digit images, has long served as a benchmark for evaluating classification algorithms. In this project, we aim to explore the effectiveness of semi-supervised learning techniques, particularly the Expectation-Maximization (EM) algorithm, in digit recognition. Additionally, we compare clustering-based approaches such as Gaussian

Mixture Model (GMM) and K-means clustering with a Random Forest classifier to assess their performance in digit recognition tasks.



2. Dataset Overview

MNIST consists of 70,000 grayscale images of handwritten digits, each measuring 28x28 pixels. The dataset is divided into 60,000 training samples and 10,000 test samples, with corresponding digit labels ranging from 0 to 9. Each pixel intensity ranges from 0 to 255, representing the darkness of the pixel.



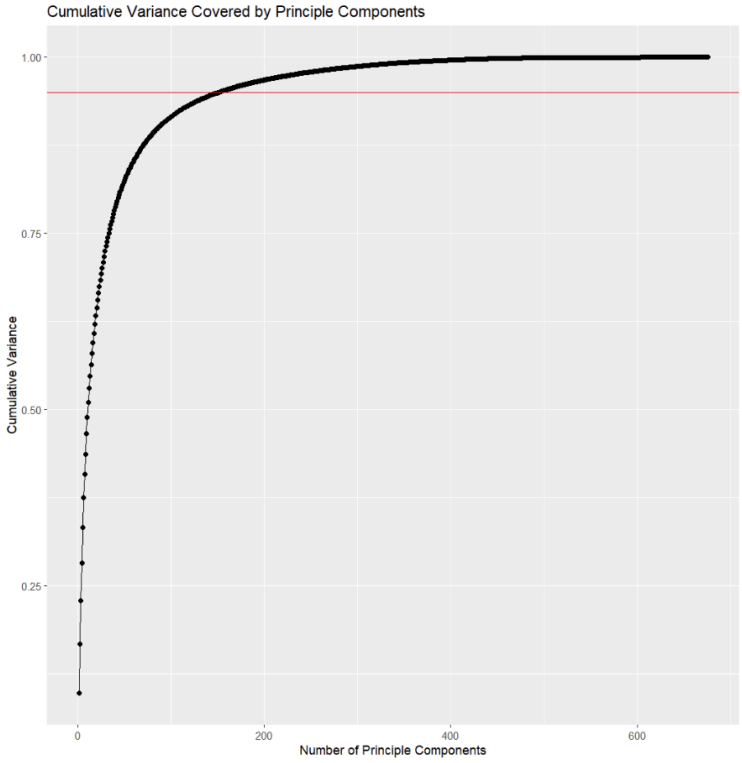
3. Methodology

3.1. Data Preprocessing

Prior to model training, the dataset undergoes preprocessing steps including normalization of pixel values to a range of [0, 1] and flattening of image matrices into vectors. This ensures uniformity in data representation across all samples.

3.2. Feature Extraction: Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is employed to reduce the dimensionality of the dataset while preserving the variance present in the original data. By selecting principal components that capture the majority of the variance, PCA facilitates more efficient computation and potentially enhances model performance.



Cumulative Variance Explained by Principal Components (PCA):

This graph illustrates the cumulative variance explained by each principal component obtained through Principal Component Analysis (PCA). It provides insights into the amount of variance retained as we increase the number of principal components. The point where the curve starts to

plateau indicates the number of components needed to capture most of the variance in the data while reducing dimensionality.

3.3. Labeling

To facilitate supervised learning, a subset of the data is manually labeled with their corresponding digit. This labeled subset serves as the initial training set, enabling the evaluation of both supervised and semi-supervised learning approaches.

3.4. Expectation-Maximization (EM) Algorithm for Semi-Supervised Learning

The Expectation-Maximization (EM) algorithm is utilized in a semi-supervised learning framework to leverage both labeled and unlabeled data. In the E-step, a Gaussian Mixture Model (GMM) estimates the posterior probability of each unlabeled image for every digit label. The M-step involves updating the parameters of the GMM based on these probabilities.

3.5. Model Training and Evaluation

The manually labeled dataset and images labeled by the EM algorithm are combined to create a comprehensive training set. A classification model is then trained using this combined dataset, enabling the evaluation of various classification techniques, including Random Forest and clustering-based methods.

3.6. Random Forest Classifier

A Random Forest classifier is trained using the preprocessed data to serve as a benchmark for comparison with clustering-based methods. Random Forest is chosen for its robustness, scalability, and ability to handle high-dimensional data.

3.7. K-means Mini-Batch Clustering

As an alternative clustering approach, K-means clustering is employed to assess its performance in digit recognition. K-means clustering partitions the data into k clusters based on similarity, offering insights into the clustering behavior of handwritten digits.

4. Results and Discussion

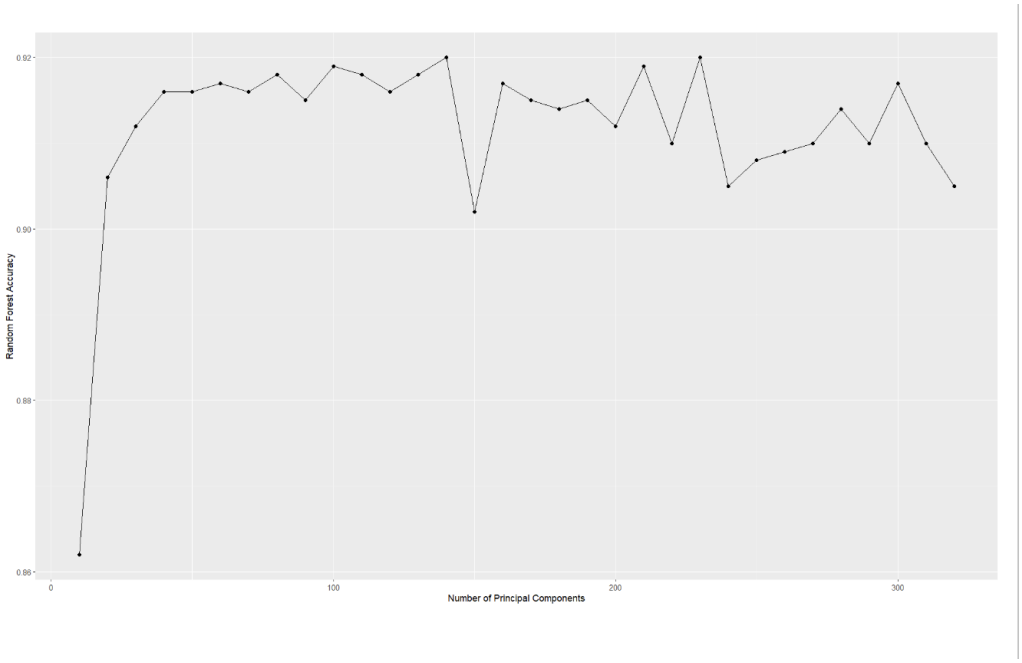
4.1. Evaluation Metrics

In evaluating the performance of different methods, we consider metrics such as accuracy. This metric provides comprehensive insights into the effectiveness of each approach in digit recognition tasks.

4.2. Performance Comparison

Results indicate that while semi-supervised learning techniques show promise, they face challenges in accurately clustering handwritten digits. Random Forest classification outperforms

clustering methods, emphasizing the importance of robust classification algorithms in digit recognition tasks.



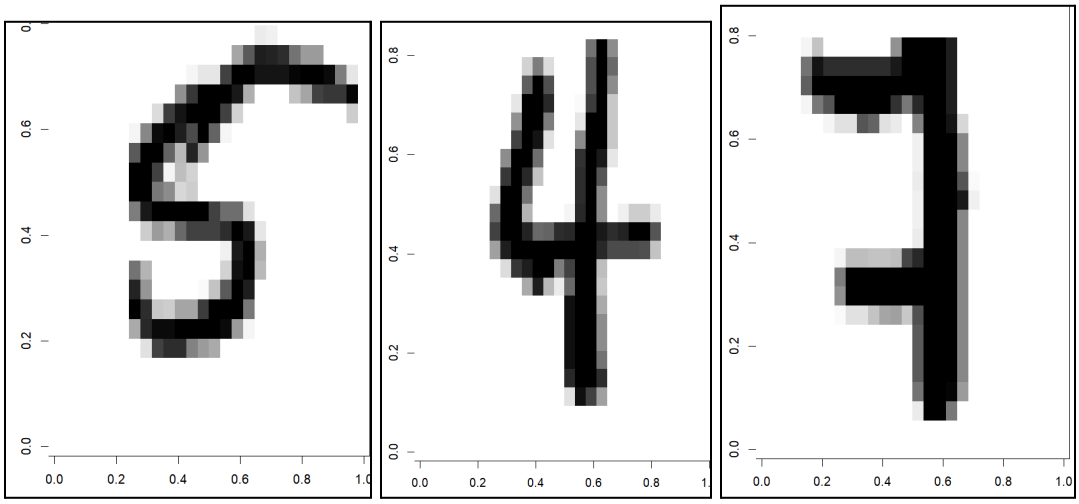
Accuracy vs. Number of Principal Components:

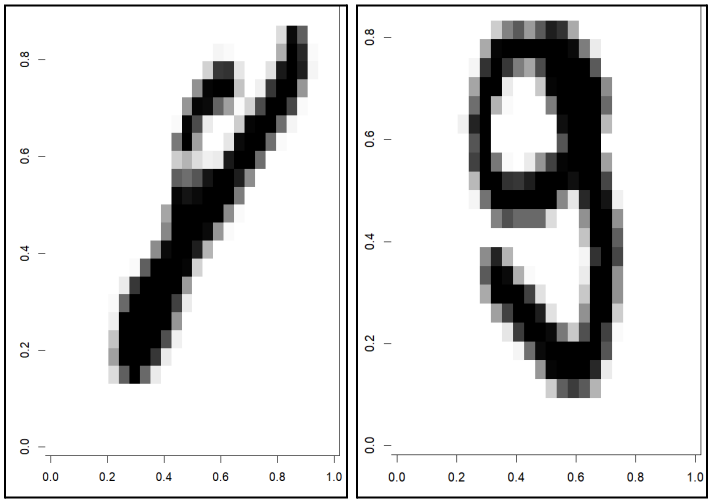
This graph demonstrates how the classification accuracy of the model varies with the number of principal components used for feature extraction in PCA. It provides valuable insights into the impact of dimensionality reduction on the model's performance, highlighting the trade-off between reducing computational complexity and preserving classification accuracy.

4.3. Limitations and Challenges

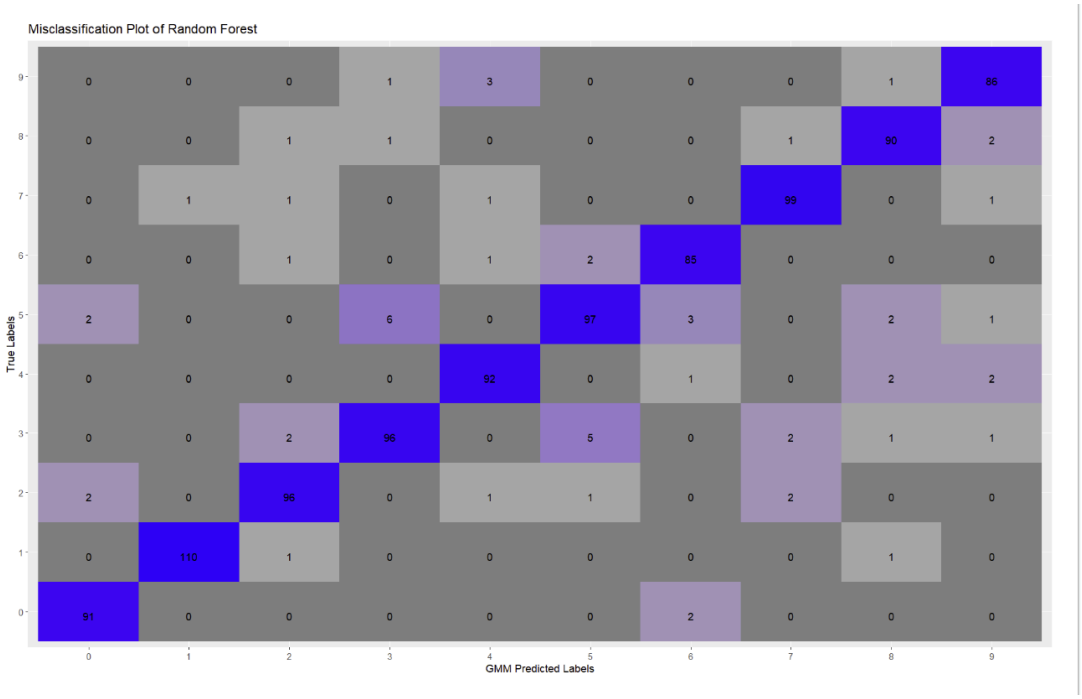
Challenges observed in clustering-based approaches include difficulties in accurately clustering digits with similar visual features and issues related to label discrepancy and misclassification.

These limitations underscore the need for further research and experimentation to enhance the performance of digit recognition models.

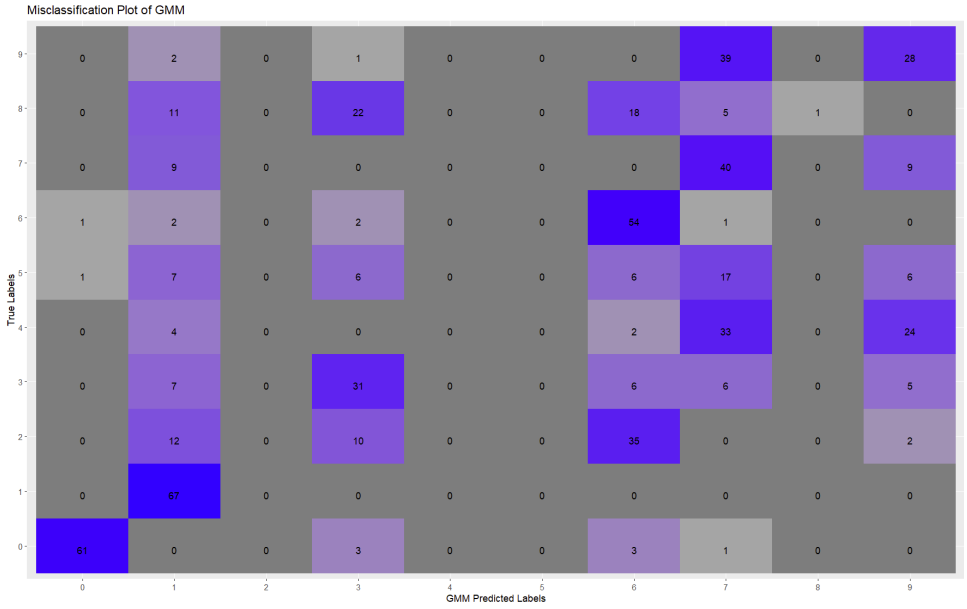




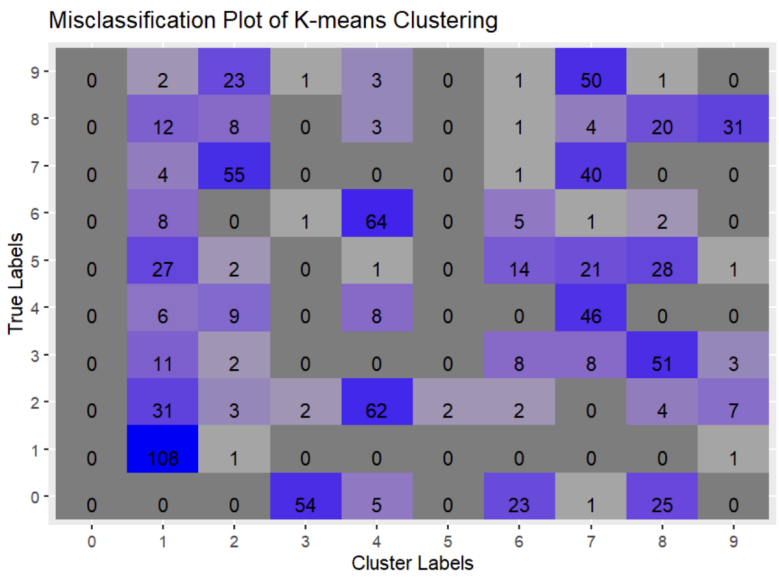
4.4. Model Performance Evaluation



Confusion Matrix for Random Forest Classifier: This matrix visually summarizes the performance of the Random Forest classifier by depicting the number of correct and incorrect predictions for each digit class. It enables us to identify patterns of misclassification and assess the model's ability to distinguish between different digits.

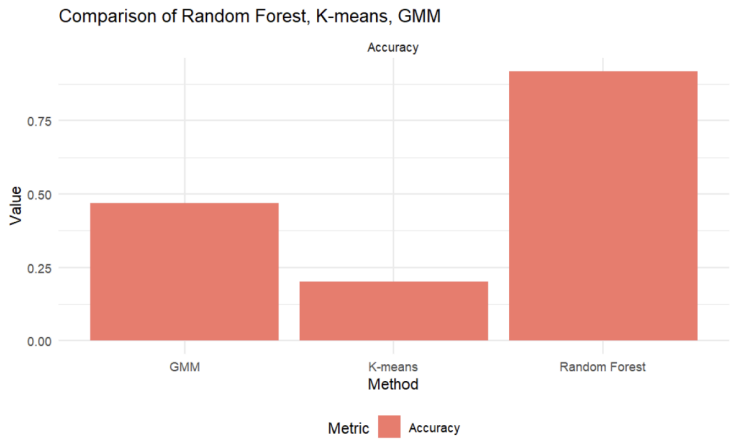


Misclassification Plot of GMM: The Gaussian Mixture Model (GMM) clustering algorithm performed poorly, achieving only a 47% accuracy rate. It struggled to establish clear clusters for digits 2, 4, and 5, resulting in misclassification or ambiguous grouping. Similarly, digit 8's cluster was barely distinguishable, indicating challenges in differentiation. These findings underscore significant limitations in the GMM clustering approach and highlight the need for improvements or alternative methodologies to enhance accuracy in digit classification tasks.



Misclassification Plot of K-means Clustering: This plot visualizes the misclassification of digits by the K-means clustering algorithm. It illustrates instances where digits are incorrectly assigned to clusters, highlighting potential limitations or ambiguities in the clustering process. Analyzing misclassification patterns can offer insights into the clustering algorithm's effectiveness in distinguishing between handwritten digits.

4.5. Accuracy Evaluation



Random Forest achieved the highest overall accuracy of 94%. However, GMM and K-means exhibited significantly lower accuracies, at 47% and 20% respectively. This disparity can be attributed to the challenges clustering methods face when applied to larger datasets like MNIST without proper data subset selection.

5. Conclusion

In conclusion, this project has offered valuable insights into the challenges and opportunities inherent in digit recognition using machine learning techniques. Through an extensive exploration of various methodologies, including semi-supervised learning and clustering algorithms, we have gained a deeper understanding of the complexities involved in accurately classifying handwritten digits.

Our results demonstrate that while semi-supervised learning approaches hold promise, they face inherent challenges in accurately clustering handwritten digits, particularly when confronted with large and diverse datasets like MNIST. Despite their potential, clustering methods such as Gaussian Mixture Model (GMM) and K-means clustering exhibited lower accuracies compared to robust classification algorithms like Random Forest.

The Random Forest classifier emerged as the top performer in our evaluation, achieving an impressive accuracy of 94%. This success underscores the importance of leveraging ensemble methods and robust classification algorithms, especially when dealing with high-dimensional data like the MNIST dataset.

However, our analysis also revealed the limitations and challenges associated with digit recognition, including difficulties in accurately clustering digits with similar visual features and issues related to label discrepancy and misclassification. These findings highlight the need for continued research and innovation in the field to overcome these challenges and improve the accuracy and efficiency of digit recognition systems.

Looking ahead, future research efforts could focus on investigating the root causes of clustering limitations, exploring novel ensemble techniques and deep learning architectures, and extending the solution to handle more complex tasks such as multi-digit number recognition. By addressing these limitations and embracing new avenues of exploration, we can continue to advance the state-of-the-art in digit recognition and contribute to the broader field of computer vision and machine learning.

6. References

1. https://www.kaggle.com/datasets/oddrationale/mnist-in-csv
2. https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/#:~:text=There%20are%20many%20ways%20for,used%20metrics%20for%20classification%20problems.
3. https://www.mathworks.com/campaigns/offers/next/choosing-the-best-machine-learning-classification-model-and-avoiding-overfitting.html
4. https://www.geeksforgeeks.org/what-is-gaussian-mixture-model-clustering-using-r/#
5. https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.11-K-Means.ipynb#scrollTo=mZ-XGUzF7gmO

6. https://medium.com/@clarice.wang/understanding-the-em-algorithm-by-examples-with-code-and-visualization-dc93657adc84
7. https://towardsdatascience.com/group-similar-image-by-using-the-gaussian-mixture-model-em-algorithm-438e9744660c
8. https://github.com/WhiskersReneeWe/EM_and_MNIST
9. https://jsks.github.io/notes/em_part2.html
10. https://github.com/kashyaparun25/Implementing-K-Means-Clustering-on-2D-and-MNIST-Data/blob/main/Implementing%20K-Means%20Clustering%20on%202D%20and%20MNIST%20Data.ipynb

7. Appendix

```r
# install tensorflow - tensorflow::install_tensorflow()
library(keras)
library(ggplot2)
library(datasets)
library(mclust)
library(ClusterR)
library(randomForest)

# Load dataset
mnist <- dataset_mnist()

# Set the desired subset sizes
train_size <- 6000
test_size <- 1000

# Subset the training data
```

```
train_indices <- sample(1:dim(mnist$train$x)[1], train_size, replace =
FALSE)
X_train <- mnist$train$x[train_indices, , ]
y_train <- mnist$train$y[train_indices]

# Subset the test data
test_indices <- sample(1:dim(mnist$test$x)[1], test_size, replace = FALSE)
X_test <- mnist$test$x[test_indices, , ]
y_test <- mnist$test$y[test_indices]

# Check the dimensions of the subsets
dim(X_train)
length(y_train)
dim(X_test)
length(y_test)

# Reshape the input data into a vector
# We know each image has 28 x 28 = 784 pixels and each pixel is a variable
X_train <- matrix(X_train, ncol = 784)
X_test <- matrix(X_test, ncol = 784)
# Observe that the training set consists of 60000 images and test set
consists
# of 10000 images
dim(X_train)
dim(X_test)

# For reference, to view the vector representing image 1 in the training
set:
X_train[1,]

# Normalize input data
# Pixel strength values range from 0 to 255 so we divide all by 255
X_train <- X_train / 255
X_test <- X_test / 255

# Convert labels to categorical vectors with 10 classes (0 through 9)
to_categorical <- function(integer_labels, num_classes = NULL) {
  if (is.null(num_classes))
    num_classes <- max(integer_labels) + 1
  out <- matrix(0, nrow = length(integer_labels), ncol = num_classes)
  rows <- seq_along(integer_labels)
  cols <- integer_labels + 1
  out[cbind(rows, cols)] <- 1
```

```r
  return(out)
}
y_train <- to_categorical(y_train, num_classes = 10)
y_test <- to_categorical(y_test, num_classes = 10)

#############################################################
# PCA

# Display original image(s) before PCA
I <- matrix(X_train[5, 784:1], nrow = 28, ncol = 28, byrow = T)
I <- apply(I, 2, rev)
image(I, col = grey(seq(1, 0, length = 784)))

# Remove constant or zero columns from the training and test data
non_zero_cols <- apply(X_train, 2, function(x) !all(x == 0))
X_train <- X_train[, non_zero_cols]
X_test <- X_test[, non_zero_cols]

pca <- prcomp(X_train)
X_train_pca <- predict(pca, X_train)
X_test_pca <- predict(pca, X_test)
# Now we have the transformed dataset

# We can use it for variable selection: select a subset of principle
# components that capture the most variance (show the most change
# between images) and only use that subset in the classification.
# This would reduce dimensionality and improve efficiency and
# performance of the classification

# pca$rotation shows the principle components PC1,...,PC784 in order
# of most impactful to least impactful
View(pca$rotation)

# Find the variance accounted for by each PC
var_explained <- pca$sdev^2 / sum(pca$sdev^2)
# Find the cumulative variance accounted for
cumulative_var <- cumsum(var_explained)
head(cumulative_var)
# So PC1 accounts for ~9% of the total variance, PC1 + PC2 account for
# ~17%, PC1 + PC2 + PC3 account for ~23%, and so on

# Plot the number of PCs (t) vs the cumulative variance
df <- data.frame(t = 1:length(cumulative_var),
```

```r
                cumulative_var = cumulative_var)
ggplot(df, aes(x = t, y = cumulative_var)) + geom_line() + geom_point() +
  geom_hline(yintercept = 0.95, color = "red") +
  labs(title = "Cumulative Variance Covered by Principle Components",
       x = "Number of Principle Components",
       y = "Cumulative Variance")
# The red line represents 90% of the total variance of the data
# So we can use a relatively small amount of PCs and still have 90% of
# the total variance

t <- which(cumulative_var >= 0.95)[1]
t
# So the first t PCs account for 90.01% of the total variance
cumulative_var[t]

# Use only the first t PCs
Xt_train_pca <- X_train_pca[, 1:t]
Xt_test_pca <- X_test_pca[, 1:t]


######################
# Random Forest Classifier

# Convert y_train from a matrix to a vector of class labels
# Subtract 1 to convert from 1-indexed to 0-indexed
train_labels <- apply(y_train, 1, which.max) - 1
test_labels <- apply(y_test, 1, which.max) - 1
# Convert to factors
class_levels <- as.character(0:9)
y_train_factor <- factor(train_labels, levels = class_levels)
y_test_factor <- factor(test_labels, levels = class_levels)

# Fit model
rf_model <- randomForest(x = X_train, y = y_train_factor)
pred_y_test <- predict(rf_model, newdata = X_test)

accuracy <- sum(pred_y_test == test_labels) / length(test_labels)
accuracy

# Confusion matrix
conf_matrix <- table(Actual = test_labels, Predicted = pred_y_test)
conf_matrix
# Confusion matrix
```

```r
confusion_matrix <- as.data.frame(table(pred_y_test, test_labels))
ggplot(data = confusion_matrix,
       mapping = aes(x = pred_y_test,
                     y = test_labels)) +
  geom_tile(aes(fill = Freq)) +
  geom_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +
  scale_fill_gradient(low = "darkgrey",
                      high = "blue",
                      trans = "log") +
  labs(title = "Misclassification Plot of Random Forest",
       x = "GMM Predicted Labels", y = "True Labels")

# Print one of the decision trees
dt1 <- getTree(rf_model, k=1, labelVar = F)
View(dt1)

# Cumulative Variance of PCs vs Test Accuracy
accuracies <- numeric()
k <- 1
for (i in seq(0.1, 1, 0.05)) {
  t <- which(cumulative_var >= i)[1]
  Xt_train_pca <- X_train_pca[, 1:t]
  Xt_test_pca <- X_test_pca[, 1:t]

  rf_model <- randomForest(x = Xt_train_pca, y = y_train_factor)
  pred_y_test <- predict(rf_model, newdata = Xt_test_pca)
  accuracy <- sum(pred_y_test == test_labels) / length(test_labels)
  accuracies[k] <- accuracy

  k <- k+1
}

df <- data.frame(cbind(seq(0.1, 1, 0.05), accuracies))
ggplot(data = df, aes(x = df[,1], y = df[,2])) + geom_line() + geom_point()

# Number of PCs vs Test Accuracy
accuracies <- numeric()
k <- 1
tmax <- which(cumulative_var >= 0.99)[1]
tmin <- which(cumulative_var >= 0.1)[1]
for (i in seq(10, tmax, 10)) {
  Xt_train_pca <- X_train_pca[, 1:i]
  Xt_test_pca <- X_test_pca[, 1:i]
```

```
  rf_model <- randomForest(x = Xt_train_pca, y = y_train_factor)
  pred_y_test <- predict(rf_model, newdata = Xt_test_pca)
  accuracy <- sum(pred_y_test == test_labels) / length(test_labels)
  accuracies[k] <- accuracy

  k <- k+1
}
df <- data.frame(cbind(seq(10, tmax, 10),
                       accuracies[1:length(seq(10, tmax, 10))]))
ggplot(data = df, aes(x = df[,1], y = df[,2])) + geom_line() + geom_point()
+
  labs(x = "Number of Principal Components", y = "Random Forest Accuracy")


#####################################
# K-means Mini-Batch Clustering

k <- 10  # Number of clusters
batch_size <- 100  # Batch size
num_init <- 10  # Number of initializations
max_iters <- 100  # Maximum number of iterations
initializer <- "kmeans++"  # Initialization method
# Number of iterations to continue after calculating the best
# within-cluster sum of squared errors
early_stop_iter <- 10

# Train k-means model
kmeans_model <- MiniBatchKmeans(
  data = X_train,
  clusters = k,
  batch_size = batch_size,
  num_init = num_init,
  max_iters = max_iters,
  initializer = initializer,
  early_stop_iter = early_stop_iter,
  verbose = TRUE
)

# Get cluster assignments for the test set
cluster_assignments_test <- predict(kmeans_model, newdata = X_test,
                                    fuzzy = FALSE)
```

```r
# Convert cluster assignments to categorical labels
cluster_labels_test <- factor(cluster_assignments_test, levels = 0:9)

# Compare k-means clustering with true labels in the test set
conf_matrix_kmeans_test <- table(cluster_labels_test, test_labels)
conf_matrix_kmeans_test

# Calculate accuracy of k-means clustering on the test set
accuracy_kmeans_test <- sum(diag(conf_matrix_kmeans_test)) /
  sum(conf_matrix_kmeans_test)
accuracy_kmeans_test

# Plot the confusion matrix
ggplot(data = as.data.frame(conf_matrix_kmeans_test),
       aes(x = cluster_labels_test, y = test_labels, fill = Freq)) +
  geom_tile() +
  geom_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +
  scale_fill_gradient(low = "darkgrey", high = "blue", trans = "log") +
  labs(title = "Misclassification Plot of K-means Clustering",
       x = "Cluster Labels", y = "True Labels")


#####################################
# GMM

# Use mclust package

# Mclust function only worked with a maximum of 600 training samples
# Possibly could only handle so much data
train_size <- 600
test_size <- 100
train_indices <- sample(1:dim(mnist$train$x)[1], train_size, replace =
FALSE)
X_train <- mnist$train$x[train_indices, , ]
y_train <- mnist$train$y[train_indices]
# Subset the test data
test_indices <- sample(1:dim(mnist$test$x)[1], test_size, replace = FALSE)
X_test <- mnist$test$x[test_indices, , ]
y_test <- mnist$test$y[test_indices]
dim(X_train)
dim(y_train)
dim(X_test)
dim(y_test)
```

```r
# Convert data matrix into vector
X_train <- matrix(X_train, ncol = 784)
X_test <- matrix(X_test, ncol = 784)
dim(X_train)
dim(X_test)
# Normalize
X_train <- X_train / 255
X_test <- X_test / 255
# Convert labels to categorical vectors with 10 classes (0 through 9)
to_categorical <- function(integer_labels, num_classes = NULL) {
  if (is.null(num_classes))
    num_classes <- max(integer_labels) + 1
  out <- matrix(0, nrow = length(integer_labels), ncol = num_classes)
  rows <- seq_along(integer_labels)
  cols <- integer_labels + 1
  out[cbind(rows, cols)] <- 1
  return(out)
}
y_train <- to_categorical(y_train, num_classes = 10)
y_test <- to_categorical(y_test, num_classes = 10)
# Remove constant or zero columns from the training and test data
non_zero_cols <- apply(X_train, 2, function(x) !all(x == 0))
X_train <- X_train[, non_zero_cols]
X_test <- X_test[, non_zero_cols]

gmm_model <- Mclust(X_train, G = 10)
summary(gmm_model)

# Convert y_train from a matrix to a vector of class labels
# Subtract 1 to convert from 1-indexed to 0-indexed
true_labels <- apply(y_train, 1, which.max) - 1

# Get the posterior probabilities for each observation
posterior_probs <- predict(gmm_model, newdata = X_train)$z

# Assign each observation to the cluster with the highest posterior
probability
assigned_clusters <- apply(posterior_probs, 1, which.max)

# Map assigned clusters to digit labels based on majority voting
assigned_labels <- rep(0, nrow(X_train))
for (cluster in 1:10) {
  cluster_indices <- which(assigned_clusters == cluster)
```

```r
  digit_labels <- true_labels[cluster_indices]
  assigned_labels[cluster_indices] <-
    as.numeric(names(which.max(table(digit_labels))))
}

# Calculate accuracy
gmm_accuracy <- sum(assigned_labels == true_labels) / length(true_labels)

# Print assigned labels and accuracy
table(assigned_labels)
print(paste("Accuracy:", gmm_accuracy))

# Confusion matrix
assigned_labels <- factor(assigned_labels, levels = 0:9)
conf_matrix <- table(Actual = true_labels,
                     Predicted = assigned_labels)
conf_matrix

confusion_matrix <- as.data.frame(table(assigned_labels, true_labels))
ggplot(data = confusion_matrix,
       mapping = aes(x = assigned_labels,
                     y = true_labels)) +
  geom_tile(aes(fill = Freq)) +
  geom_text(aes(label = sprintf("%1.0f", Freq)), vjust = 1) +
  scale_fill_gradient(low = "darkgrey",
                      high = "blue",
                      trans = "log") +
  labs(title = "Misclassification Plot of GMM",
       x = "GMM Predicted Labels", y = "True Labels")


#########################################################
# Comparing GMM, Random Forest, and Kmeans Accuracies
# Random Forest Comparison
rf_comparison <- data.frame(
  Method = c("Random Forest"),
  Metric = c("Accuracy"),
  Value = c(accuracy)
)
# K-means Comparison
kmeans_comparison <- data.frame(
  Method = c("K-means"),
  Metric = c("Accuracy"),
```

```
  Value = c(accuracy_kmeans_test)
)
# GMM Comparison
gmm_comparison <-data.frame(
  Method = c("GMM"),
  Metric = c("Accuracy"),
  Value = c(gmm_accuracy)
)
# Combine both comparisons
comparison_data <- rbind(rf_comparison, kmeans_comparison, gmm_comparison)
# Plot
ggplot(comparison_data, aes(x = Method, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  facet_wrap(~ Metric, scales = "free_y") +
  labs(
    title = "Comparison of Random Forest, K-means, GMM",
    x = "Method",
    y = "Value",
    fill = "Metric"
  ) +
  theme(legend.position = "bottom")
```