Spring 2023
CS 464 INTRODUCTION TO DATA SCIENCE

Yagiz Yaman

2023-05-09

Dimensionality Reduction and Visualization Project

Programming Languages Used

R

For splitting the dataset into train/test, centering the dataset (Question 1.1) and applying t-SNE (Question 3), R programming language is used.

Python

For PCA, Isomap and classification tasks (Question 1.2, 1.3, 1.4 & Question 2), Python is used.

Programming Environment

The codes are written in an .rmd file as it is convenient to use different programming languages together.

Assignment Description

In this assignment, the goal is to develop a handwritten digit recognition system. The input to the system will be a digitized image of a digit and the output will be the type of the digit {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. The digits.txt contains a matrix with 5,000 rows where each row corresponds to a pattern and 400 columns and the labels.txt contains a vector with length 5000 where each value shows the class for the corresponding pattern for example 1,2,3 etc. Each pattern is a digitized 20 X 20 image of a digit (represented in column-major order as a 400-dimensional vector). Download the digit data set and divide it into two subsets by randomly selecting half of the patterns from each class for training and the remaining patterns for testing. Then, perform the classification experiments described below. Use the same subsets for the rest of the assignment. Do not forget to use separate subsets for training and testing.

A quadratic Gaussian classifier models each class with a Gaussian distribution. Gaussian can be considered as a model where the feature vectors for a given class are continuous-valued, randomly corrupted versions of a single typical or prototype vector. The Gaussian density for the feature vector $x \in R^d$ is defined as,

$$p(x|\mu,\sum)=\frac{1}{(2\pi)^{d/2}*\left|\sum\right|^{1/2}}*e^{[-\frac{1}{2}*(x-\mu)^T*\left|\sum\right|^{-1}(x-\mu)]}$$

where d is the dimension of x, and μ and $\sum$ are the mean vector and the covariance matrix, respectively.

The training procedure involves fitting a Gaussian $p_c(x|\mu_c, \sum_c)$ to the corresponding subset of the training data for each class c = 1,. . . ,C where C is the number of classes. Fitting can be done by using the maximum likelihood estimates of the mean vector and the covariance matrix,

$$\hat{\mu} = \frac{1}{n} * \sum_{i=1}^{n} x_i$$

$$\hat{\sum} = \frac{1}{n} * \sum_{i=1}^{n} (x_i - \hat{\mu}) * (x_i - \hat{\mu})^T$$

using the samples for each class. In the formulas above, n represents the number of samples that belong to a particular class. Estimation should be done separately by using the subset of samples for each class.

Once the Gaussian densities are learned for all classes, during the classification process, a pattern can be assigned to the class c^* that gives the highest probability for that patternâ€™s feature vector x as,

$$c^* = \underset{c=1,\ldots,C}{argmax}\, P_c(x|\mu_c, \sum\nolimits_c)$$

Given a data set with known class labels, the performance of the classifier can be evaluated by comparing the predicted class by the classifier to the true class known for each sample. The quantitative performance can be summarized using the classification error that is computed as the ratio of the number of wrongly predicted samples to the total number of samples.

QUESTION 1 [35 pts]

In this question, you will use principal components analysis (PCA) to project the 400-dimensional data onto lower dimensional subspaces to observe the effect of dimensionality on the performance of the Gaussian classifier.

Question 1.1

First, center the data by subtracting the mean of the whole data from each sample.

Answer 1.1

Load the datasets and center the data.

```
digits <- read.table("C:/Users/yagiz/Desktop/4-2/GE-461/DimensionalityReduction/digits/digits.txt")
labels <- read.table("C:/Users/yagiz/Desktop/4-2/GE-461/DimensionalityReduction/digits/labels.txt")

centered_digits <- sweep(digits, 2, colMeans(digits), "-")
```

We **centered the data**. Now we are going to analyze the data and then split into train & test.

```
digits <- as.data.table(digits)
labels <- as.data.table(labels)
digits[, label := labels[, V1]]
digits[, label := labels[, V1]]

sort(unique(digits[, label]))
```

```
##  [1] 0 1 2 3 4 5 6 7 8 9
```

Now we have centered the data. Examine the dataset. As seen we have 10 labels. Now we are going to check their distributions.

```
digits[, .N, .(label)][order(label)]
```

```
##     label   N
##  1:     0 460
##  2:     1 571
##  3:     2 530
##  4:     3 500
##  5:     4 500
##  6:     5 456
##  7:     6 462
##  8:     7 512
##  9:     8 489
## 10:     9 520
```

As seen they are nearly equal. Now we are going to create train_data & test_data by dividing digits into two subsets by randomly selecting half of the patterns from each class for training and the remaining patterns for testing.

We are going to divide datasets for each label and then randomly divide them into two.

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

data_train <- digits[0L]
data_test <- digits[0L]
for (i in 0:9){
  digits_sub = digits[label == i]
  nrow_sub = round(nrow(digits_sub)/2)
  train_rows = sample(nrow(digits_sub), size = nrow_sub)


  train_table <- digits_sub[train_rows, ]

  test_rows <- setdiff(1:nrow(digits_sub), train_rows)
  test_table <- digits_sub[test_rows, ]

  data_train <- rbind(data_train, train_table)
  data_test <- rbind(data_test, test_table)
}

labels_train <- data_train[, .(label)]
labels_test <- data_test[, .(label)]

data_train[, label := NULL]
data_test[, label := NULL]


# write these data to csv to read from python.
fwrite(data_train, "C:/Users/yagiz/Desktop/4-2/GE-461/DimensionalityReduction/data/train_data.csv")
fwrite(data_test, "C:/Users/yagiz/Desktop/4-2/GE-461/DimensionalityReduction/data/test_data.csv")
fwrite(labels_train, "C:/Users/yagiz/Desktop/4-2/GE-461/DimensionalityReduction/data/train_labels.csv")
fwrite(labels_test, "C:/Users/yagiz/Desktop/4-2/GE-461/DimensionalityReduction/data/test_labels.csv")
```

```
train_data = pd.read_csv("C:/Users/yagiz/Desktop/4-2/GE-461/DimensionalityReduction/data/train_data.csv
train_data = train_data.values
train_data = np.reshape(train_data, (2500, 400))

test_data = pd.read_csv("C:/Users/yagiz/Desktop/4-2/GE-461/DimensionalityReduction/data/test_data.csv",
test_data = test_data.values
test_data = np.reshape(test_data, (2500, 400))

train_labels = pd.read_csv("C:/Users/yagiz/Desktop/4-2/GE-461/DimensionalityReduction/data/train_labels
train_labels = train_labels.values
train_labels = np.reshape(train_labels, (2500,))

test_labels = pd.read_csv("C:/Users/yagiz/Desktop/4-2/GE-461/DimensionalityReduction/data/test_labels.c
test_labels = test_labels.values
test_labels = np.reshape(test_labels, (2500,))
```

Question 1.2

Then, use PCA to obtain a new set of bases (use the training data set, i.e., 2,500 samples for PCA). Plot the eigenvalues in descending order. How many components (subspace dimension) would you choose by just looking at this plot?

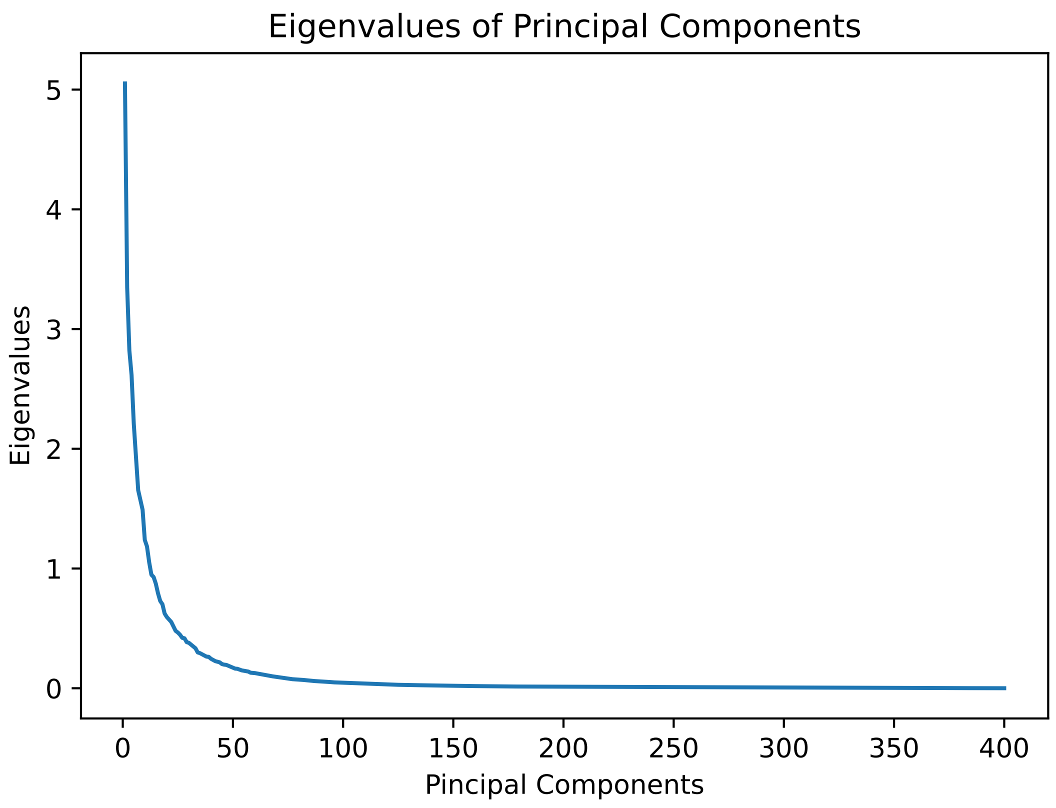
Answer 1.2

As scikit learn already centers data, I have used non-centered data for the answer.

```
plt.clf()
pca = PCA(n_components=400)
pca.fit(train_data)

# order the eigenvalues
```

```
## PCA(n_components=400)
```

```
descending_eigenvalues = np.sort(pca.explained_variance_)[::-1]

plt.plot(range(1, 401), descending_eigenvalues)
plt.xlabel('Pincipal Components')
plt.ylabel('Eigenvalues')
plt.title('Eigenvalues of Principal Components')
plt.show()
```



I would choose around 50 components as it looks like an elbow in the graph. After 50, the eigenvalues are quite less and it means that those principal components do not represent the significant amount of the variability in the data.

Question 1.3

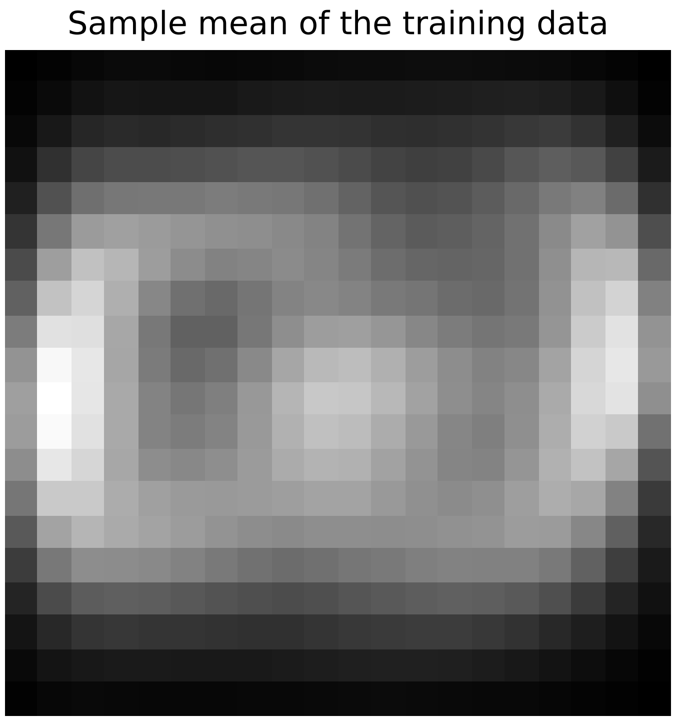
Display the sample mean for the whole training data set as an image (using samples for all classes together). Also display the bases (eigenvectors) that you chose as images and discuss the results with respect to your expectations.

Answer 1.3

```
plt.clf()
# Display the sample mean as an image
train_mean = np.mean(train_data, axis=0)
image_sample_mean = np.reshape(train_mean, (20, 20))


plt.imshow(image_sample_mean, cmap='gray')
plt.title('Sample mean of the training data')
plt.axis('off')
```

```
## (-0.5, 19.5, 19.5, -0.5)
```

```
plt.show()
```

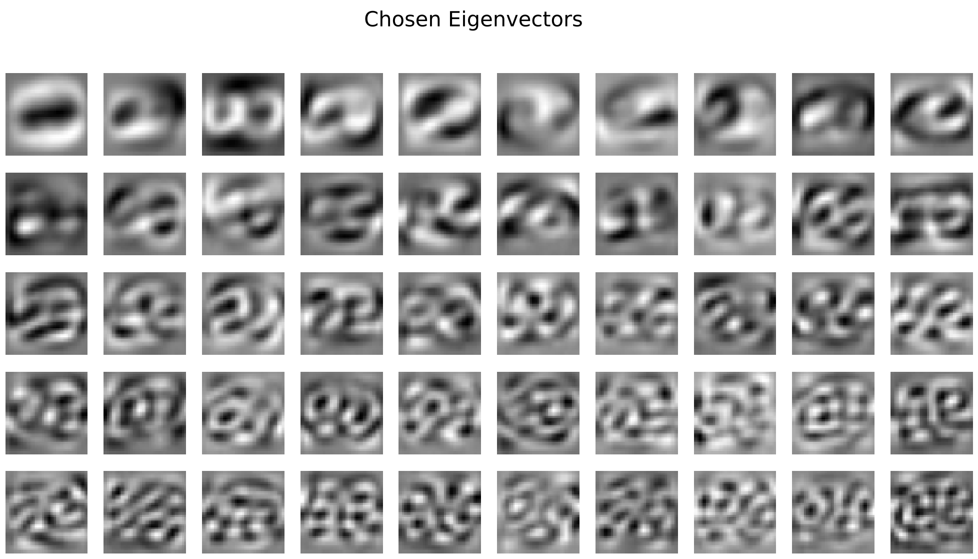


The shape of the sample mean has a curved shape. Considering most numbers are curvy, it is expected. It looks like 8.

```
plt.clf()
# display eigenvectors as an image
eigenvectors = pca.components_.reshape(-1, 20, 20)

chosen_eigenvectors = eigenvectors[:50]

fig, axes = plt.subplots(5, 10, figsize=(10, 5))
for i, ax in enumerate(axes.flat):
    ax.imshow(chosen_eigenvectors[i], cmap='gray')
    ax.axis('off')
```

```
## <matplotlib.image.AxesImage object at 0x000001E95D63B730>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D63BE80>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D63BA60>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D639600>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D670190>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6704C0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6707F0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D670B20>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D670E50>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D671180>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6714B0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6717E0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D671B10>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D671E40>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D672170>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6724A0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6727D0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D672B00>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D672E30>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D673160>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D673490>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6737C0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D673AF0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D673E20>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AC190>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AC4C0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AC7F0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6ACB20>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6ACE50>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AD180>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AD4B0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AD7E0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6ADB40>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6ADE70>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AE1A0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AE4D0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AE800>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AEB30>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AEE60>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AF190>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AF4C0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AF7F0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AFB20>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6AFE50>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6EC1C0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6EC4F0>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6EC820>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6ECB50>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6ECE80>
## (-0.5, 19.5, 19.5, -0.5)
## <matplotlib.image.AxesImage object at 0x000001E95D6ED1B0>
## (-0.5, 19.5, 19.5, -0.5)
```

```
plt.suptitle('Chosen Eigenvectors')
plt.show()
```



Again, the eigenvectors looks quite curvy. They also look like 8.

Question 1.3

Choose different subspaces with dimensions between 1 and 200 (choose at least 20 different subspace dimensions, the more the better), and project the data (project both the training data and the test data using the transformation matrix estimated from the training data) onto these subspaces. Train a Gaussian classifier using data in each subspace (do not forget to use half of the data for training and the remaining half for testing).

Answer 1.3

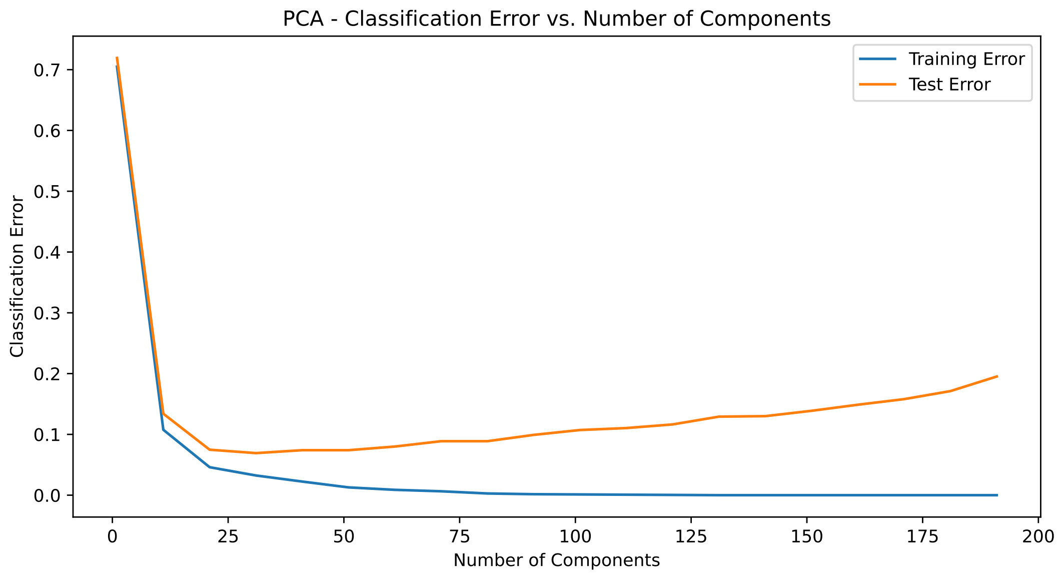
```
subspace_dimensions = range(1, 201, 10)

train_error_pca = []
test_error_pca = []

for i in subspace_dimensions:
    # Perform PCA
    pca = PCA(n_components=i)
    pca.fit(train_data)

    # Project the training and test data onto the subspace
    train_data_projected = pca.transform(train_data)
    test_data_projected = pca.transform(test_data)

    classifier = QuadraticDiscriminantAnalysis()
    classifier.fit(train_data_projected, train_labels)

    # Train the classifier
    classifier = QuadraticDiscriminantAnalysis()
    classifier.fit(train_data_projected, train_labels)

    # Predict labels
    train_predictions = classifier.predict(train_data_projected)
    test_predictions = classifier.predict(test_data_projected)

    # Calculate errors
    train_err = 1-accuracy_score(train_predictions, train_labels)
    test_err = 1-accuracy_score(test_predictions, test_labels)


    train_error_pca.append(train_err)
    test_error_pca.append(test_err)
```

Question 1.4

Plot classification error vs. the number of components used for each subspace, and discuss your results. Compute the classification error for both the training set and the test set (training is always done using the training set), and provide two plots.

Answer 1.4

```
plt.clf()
plt.plot(subspace_dimensions, train_error_pca, label='Training Error')
plt.plot(subspace_dimensions, test_error_pca, label='Test Error')
plt.xlabel('Number of Components')
plt.ylabel('Classification Error')
plt.title('PCA - Classification Error vs. Number of Components')
plt.legend()
plt.show()
```



As seen when n_components is increased, the classification error is decreased continuously for training data, but it behaves differently for test data. For test data, the classification error is decreased until around number of components being equal to 23 and then it continuously increases. One possible reason for this situation is overfitting. The additional components may capture the noise in the training data and that's why, it would be less effective in an unseen test data.

QUESTION 2 [35 points]

In this question, you will use Isomap (J. B. Tenenbaum, V. de Silva, J. C. Langford, A Global Geometric Framework for Nonlinear Dimensionality Reduction,“. Science, vol. 290, pp:2319- 2323, 2000) to map the 400-dimensional data onto lower dimensional manifolds.

Question 2.1

Use Isomap to obtain low-dimensional embeddings of the digits data. Note that you need to use the full data set, i.e., 5,000 patterns, but you may still have several patterns that were not embedded. This is a common observation in many techniques that are based on neighborhood graphs where the embedding implementation only uses the largest connected component of the neighborhood graph and ignores the patterns belonging to other components.

Answer 2.1

Apply Isomap for dimensionality reduction on the digits dataset.

```
digits = np.loadtxt("C:/Users/yagiz/Desktop/4-2/GE-461/DimensionalityReduction/digits/digits.txt")
labels = np.loadtxt("C:/Users/yagiz/Desktop/4-2/GE-461/DimensionalityReduction/digits/labels.txt")
```

```
from sklearn.manifold import Isomap

isomap = Isomap(n_components=2) # apply with 2 now

isomap.fit(digits)
```

```
## Isomap()
```

```
embeddings = isomap.transform(digits)
```

Question 2.2

Choose dimensions between 1 and 200 (choose at least 20 different dimensions, the more the better) and train a Gaussian classifier for each dimensionality (do not forget to use half of the data for training and the remaining half for testing).

Answer 2.2

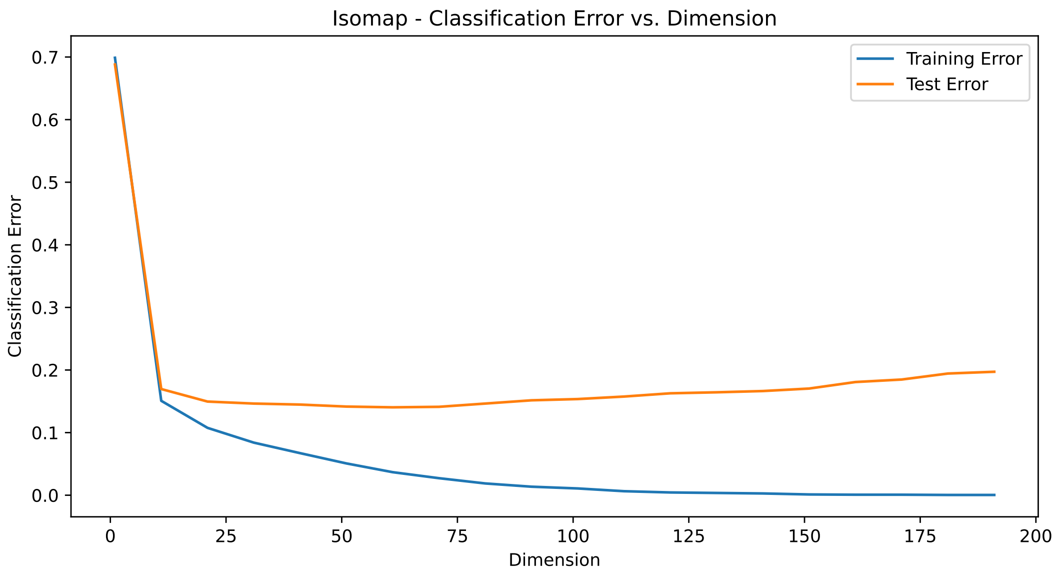
```

subspace_dimensions = range(1, 201, 10)

train_error_isomap = []
test_error_isomap = []


for i in subspace_dimensions:

    isomap = Isomap(n_components=i)

    # Fit the Isomap model to the training data
    isomap.fit(train_data)

    # Transform the training and test data to obtain the low-dimensional embeddings
    train_data_embedded = isomap.transform(train_data)
    test_data_embedded = isomap.transform(test_data)
    # Train the classifier
    classifier = QuadraticDiscriminantAnalysis()
    classifier.fit(train_data_embedded, train_labels)

    # Predict labels
    train_predictions = classifier.predict(train_data_embedded)
    test_predictions = classifier.predict(test_data_embedded)

    # Calculate classification errors
    train_err = 1 - accuracy_score(train_predictions, train_labels)
    test_err = 1 - accuracy_score(test_predictions, test_labels)

    train_error_isomap.append(train_err)
    test_error_isomap.append(test_err)
```

Question 2.3

Plot classification error vs. dimension, and discuss your results. Compute the classification error for both the training set and the test set (training is always done using the training set), and provide two plots. The discussion should include the setting (particular choice for the parameters) for Isomap, the effect of dimensionality on the classification error, and comparison of the Isomap results with the PCA results.

Answer 2.3

```
plt.clf()
plt.plot(subspace_dimensions, train_error_isomap, label='Training Error')
plt.plot(subspace_dimensions, test_error_isomap, label='Test Error')
plt.xlabel('Dimension')
plt.ylabel('Classification Error')
plt.title('Isomap - Classification Error vs. Dimension')
plt.legend()
plt.show()
```



As seen when n_components is increased, the classification error is decreased continuously for training data, but it behaves differently for test data. For test data, the classification error is decreased until around dimension is being equal to 23 and then it starts to slowly increase. Again, it may be because of overfitting. With the increase in the dimension, the model may capture the noise in the train data and that makes the model work poorly in an unseen data.

```
print(f'Mean Train Error (PCA): {np.mean(train_error_pca)}\n Mean Train Error (Isomap): {np.mean(train_
```

```
## Mean Train Error (PCA): 0.047400000000000005
##  Mean Train Error (Isomap): 0.06436
##  Mean Test Error (PCA): 0.14536
##  Mean Test Error (Isomap): 0.18759999999999996
```

As seen PCA works better. The possible reason is that PCA outperforms Isomap in capturing linear relationships. Probably there are linear relationships in the dataset.

QUESTION 3 [30 points]

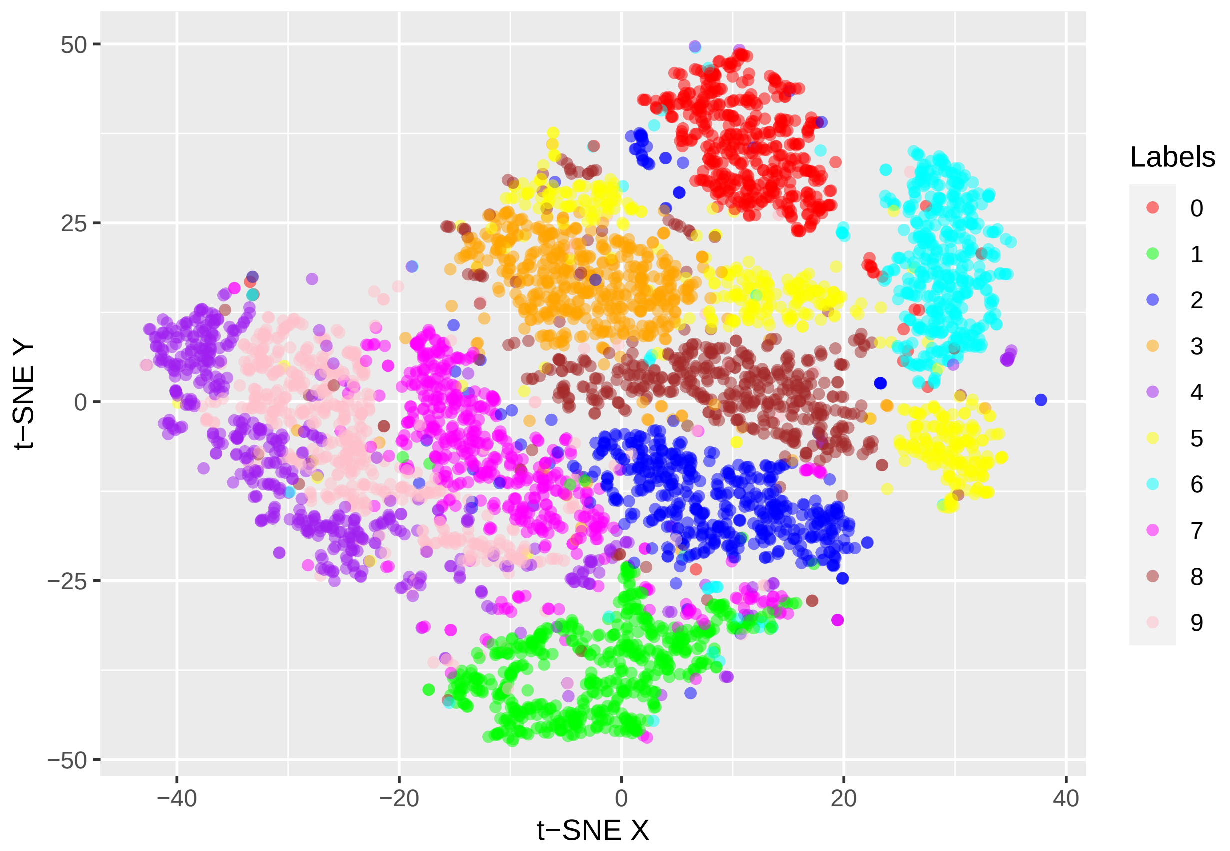
In this question, you will use dimensionality reduction particularly designed for visualization of high-dimensional data sets. In particular, you are asked to use t-SNE (L. J. P. van der Maaten and G. E. Hinton, “Visualizing High-Dimensional Data Using t-SNE,”. Journal of Machine Learning Research, vol. 9, pp:2579-2605, November 2008) for mapping the digit data set to two dimensions. Compute the resulting mapping for the whole data set, and present the scatter of the samples together with their class information. Discuss the setup that you used (e.g., parameters needed for initialization, iterations, or stopping, etc), and comment on the resulting visualizations.

Answer 3

We are going to apply t-SNE with default values and then we are going to change parameters and observe their effects.

```
set.seed(123)

my_digits <- as.data.table(digits)
```

```
tsne <- Rtsne(my_digits[, -c("label")], perplexity = 30, max_iter = 1000, eta=200)

tsne_dt <- data.table(tsne_x = tsne$Y[, 1], tsne_y = tsne$Y[, 2])
my_digits[, label := as.factor(label)]

my_palette <- colorRampPalette(colors = c("red", "green", "blue", "orange",
            "purple", "yellow", "cyan", "magenta", "brown", "pink"))(10)


ggplot(tsne_dt, aes(x= tsne_x, y= tsne_y, color=my_digits[,label])) + geom_point(alpha=0.5) +
  scale_color_manual(values = my_palette)+
  labs(x = "t-SNE X", y = "t-SNE Y", color = "Labels")
```

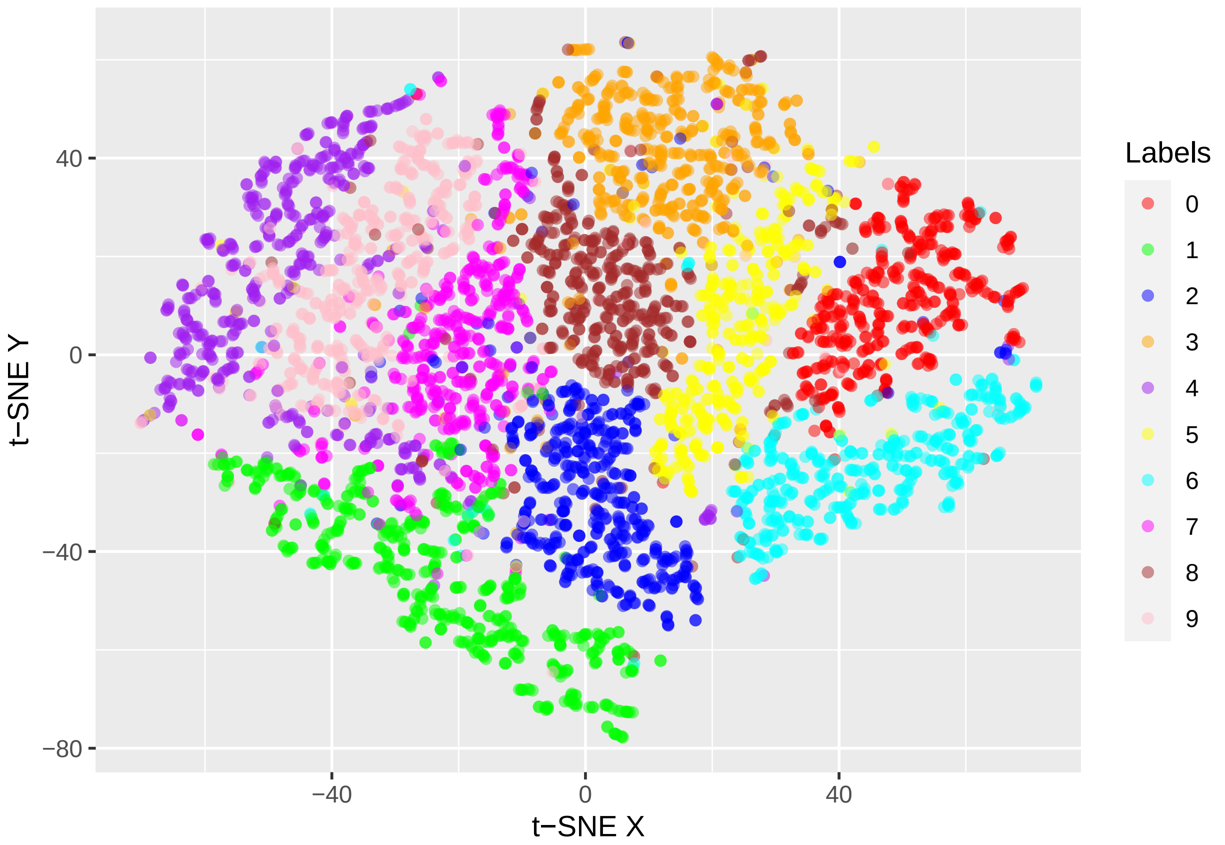


The 2-D graph shows that label 5 has not been clustered properly. Except label 5, the labels seem to be clustered properly.

Now I am going to change parameters and observe the effects on the reduction. Start with perplexity.

```
set.seed(123)

my_digits <- as.data.table(digits)

tsne <- Rtsne(my_digits[, -c("label")], perplexity = 5, max_iter = 1000, eta=200)

tsne_dt <- data.table(tsne_x = tsne$Y[, 1], tsne_y = tsne$Y[, 2])
my_digits[, label := as.factor(label)]
```

```
my_palette <- colorRampPalette(colors = c("red", "green", "blue", "orange",
            "purple", "yellow", "cyan", "magenta", "brown", "pink"))(10)


ggplot(tsne_dt, aes(x= tsne_x, y= tsne_y, color=my_digits[,label])) + geom_point(alpha=0.5) +
  scale_color_manual(values = my_palette)+
  labs(x = "t-SNE X", y = "t-SNE Y", color = "Labels")
```

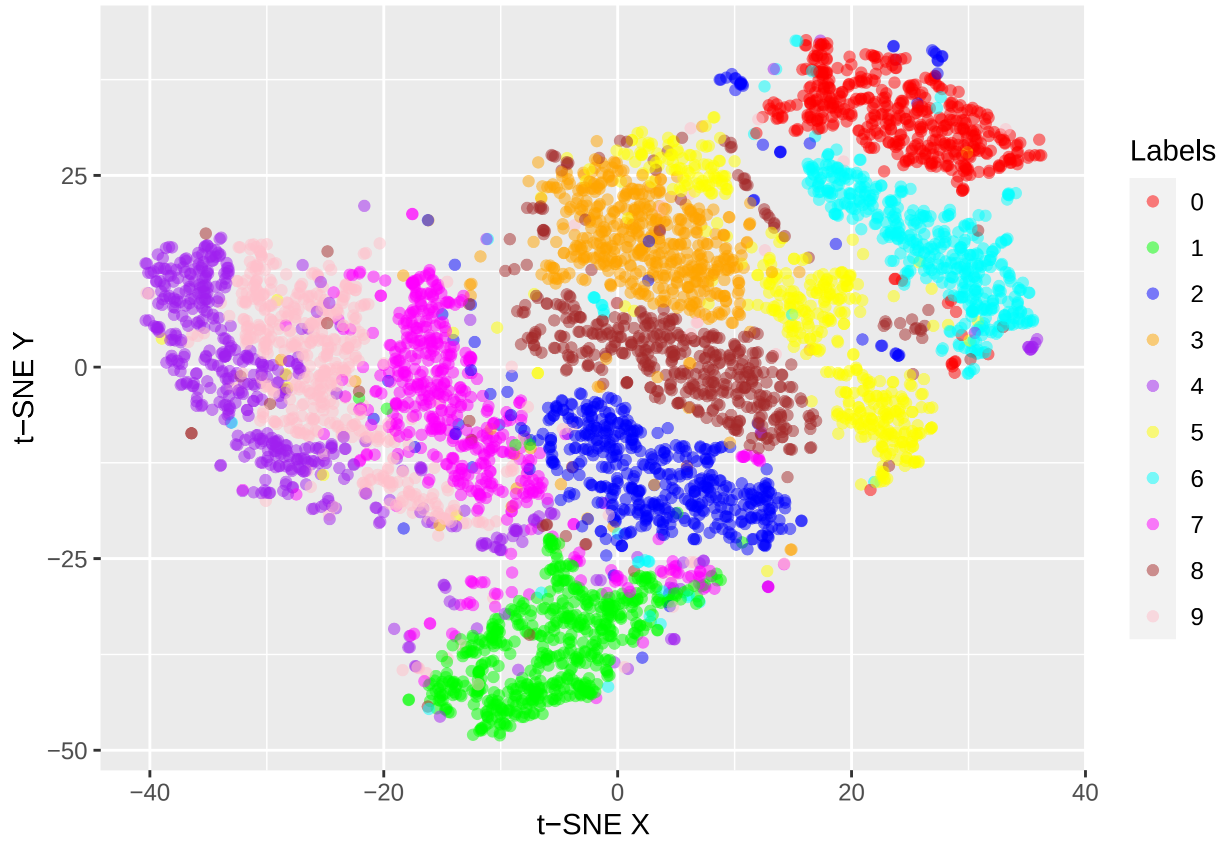


When we lower **perplexity** to 5, the t-SNE detects narrower clusters compared to perplexity being equal to 50. Also, the graph is now less dense. This is an expected result.

Learning rate is represented by eta in R. Let's increase learning rate to 1000 from 200 and observe the graph.

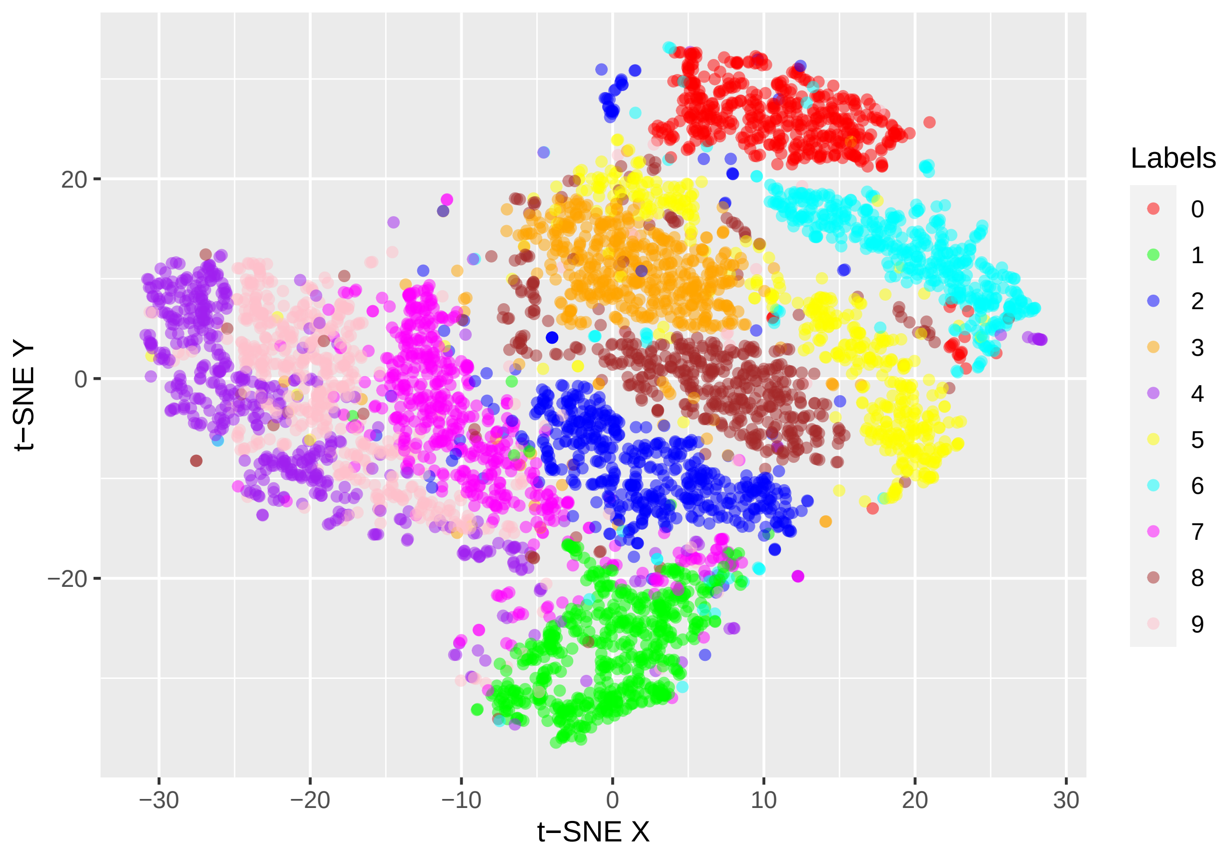
```
set.seed(123)

my_digits <- as.data.table(digits)

tsne <- Rtsne(my_digits[, -c("label")], perplexity = 50, max_iter = 1000, eta=1000)

tsne_dt <- data.table(tsne_x = tsne$Y[, 1], tsne_y = tsne$Y[, 2])
my_digits[, label := as.factor(label)]

my_palette <- colorRampPalette(colors = c("red", "green", "blue", "orange",
            "purple", "yellow", "cyan", "magenta", "brown", "pink"))(10)
```

```
ggplot(tsne_dt, aes(x= tsne_x, y= tsne_y, color=my_digits[,label])) + geom_point(alpha=0.5) +
  scale_color_manual(values = my_palette)+
  labs(x = "t-SNE X", y = "t-SNE Y", color = "Labels")
```



When learning rate is increased, the algorithm works quite faster. The graph does not change drastically from the default one.

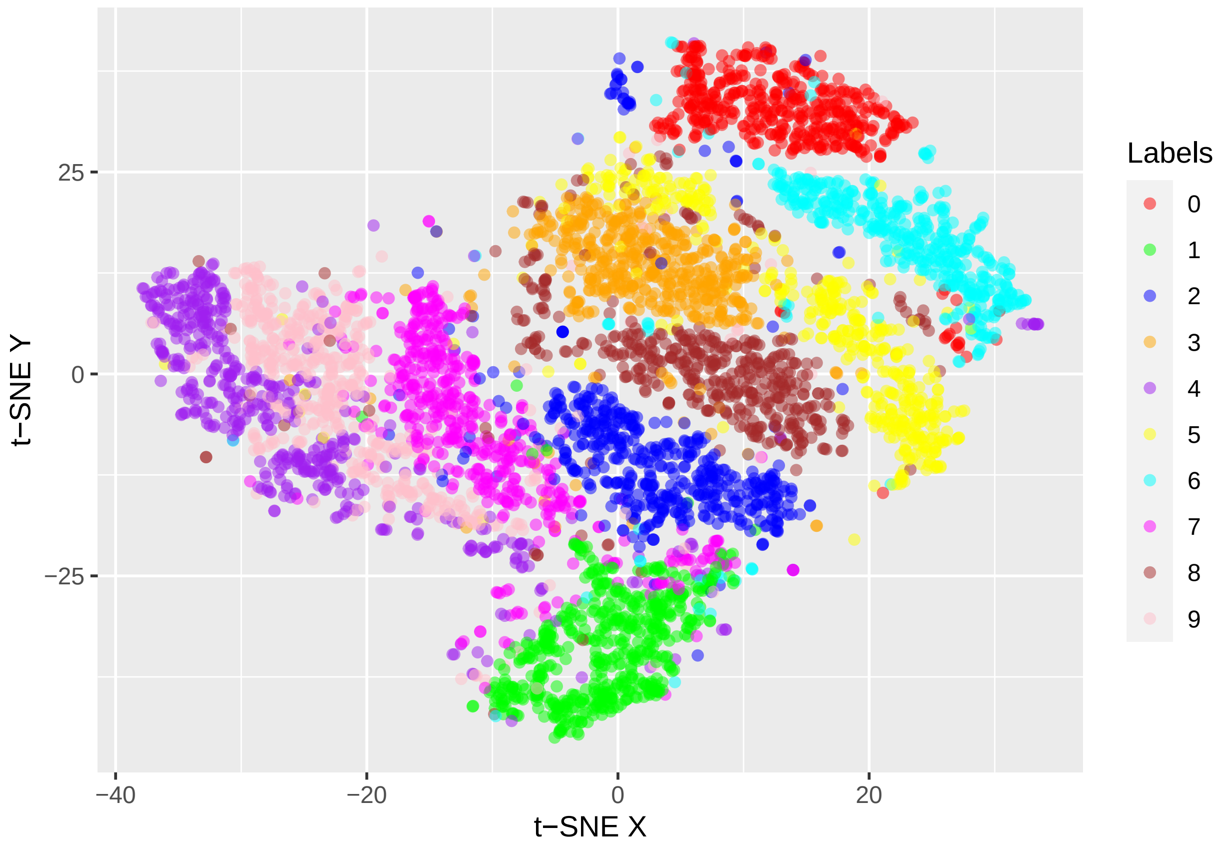
```
set.seed(123)

my_digits <- as.data.table(digits)

tsne <- Rtsne(my_digits[, -c("label")], perplexity = 50, max_iter = 1000, eta=50)

tsne_dt <- data.table(tsne_x = tsne$Y[, 1], tsne_y = tsne$Y[, 2])
my_digits[, label := as.factor(label)]

my_palette <- colorRampPalette(colors = c("red", "green", "blue", "orange",
            "purple", "yellow", "cyan", "magenta", "brown", "pink"))(10)


ggplot(tsne_dt, aes(x= tsne_x, y= tsne_y, color=my_digits[,label])) + geom_point(alpha=0.5) +
  scale_color_manual(values = my_palette)+
  labs(x = "t-SNE X", y = "t-SNE Y", color = "Labels")
```



With lower learning rate, clusters are a little bit wider.

Now i am going to observe the effect of **number of iterations**.

```
set.seed(123)

my_digits <- as.data.table(digits)

tsne <- Rtsne(my_digits[, -c("label")], perplexity = 50, max_iter = 2000, eta=50)

tsne_dt <- data.table(tsne_x = tsne$Y[, 1], tsne_y = tsne$Y[, 2])
my_digits[, label := as.factor(label)]

my_palette <- colorRampPalette(colors = c("red", "green", "blue", "orange",
            "purple", "yellow", "cyan", "magenta", "brown", "pink"))(10)


ggplot(tsne_dt, aes(x= tsne_x, y= tsne_y, color=my_digits[,label])) + geom_point(alpha=0.5) +
  scale_color_manual(values = my_palette)+
  labs(x = "t-SNE X", y = "t-SNE Y", color = "Labels")
```



When we decrease number of iterations, the clustering becomes less accurate.

References for Tools Used

For R;

- Dowle, M., & Srinivasan, A. (2020). data.table: Extension of 'data.frame'. R package version 1.14.2. Retrieved from https://CRAN.R-project.org/package=data.table
- Krijthe, J. H. (2015). Rtsne: T-Distributed Stochastic Neighbor Embedding using Barnes-Hut Implementation. R package version 0.15. Retrieved from https://CRAN.R-project.org/package=Rtsne
- Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., & Woo, K. (2021). ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics. R package version 3.3.5. Retrieved from https://CRAN.R-project.org/package=ggplot2

For Python;

- Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O.,Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830. https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html
- Hunter, J.D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95. https://doi.org/10.1109/MCSE.2007.55

- McKinney, W. (2010). Data structures for statistical computing in Python. In Proceedings of the 9th Python in Science Conference, 51-56. https://conference.scipy.org/proceedings/scipy2010/mckinney.html