REPUBLIC OF TURKEY
ADANA ALPARSLAN TÜRKEŞ SCIENCE AND TECHNOLOGY UNIVERSITY

FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

T-SNE AND 2D LDA

ÖMER HATAY
BACHELOR'S DEGREE

ADANA 2024

REPUBLIC OF TURKEY
ADANA ALPARSLAN TÜRKEŞ SCIENCE AND TECHNOLOGY UNIVERSITY

FACULTY OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

T-SNE AND 2D LDA

ÖMER HATAY
BACHELOR'S DEGREE

SUPERVISOR
ASSOC. PROF. DR. HÜSEYİN AFŞER

ADANA 2024

ABSTRACT

T-SNE AND 2D LDA

Ömer HATAY
Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Hüseyin AFŞER
01 2024, 28 pages

In this graduation project, I worked on image processing, using the python programming language. The MNIST data set is great for image processing. The MNIST database is a large database of handwritten digits. This database consists of the MNIST dataset of 70,000 images of small numbers handwritten by high school students and US Census Bureau employees. Each image is labeled with the digit it represents.

In the MNIST studied, operations such as dimensionality reduction and classification were carried out using the PyCharm editor. Displaying MNIST images in 2 dimensions with t-SNE and classifying MNIST images by reducing them to 2 dimensions with linear discriminant analysis were performed and results and code shown. The codes of the studies available at appendix section.

Keywords: MNIST, t-SNE, scikit-learn(manifold), 2-D LDA, classification, dimension reduction

Contents

NOMENCLATURE

MNIST	: Modified National Institute of Standards and Technology
t-SNE	: t-Distributed Stochastic Neighbour Embedding
LDA	: Linear Discriminant Analysis
SGD	: Stochastic Gradient Descent
FPR	: False Positive Rate
ROC	: Receiver Operating Characteristic
AUC	: Area Under the Curve
OvR	: One versus the Rest
OvO	: One versus One
KL	: Kullback-Leibler
SNE	: Stochastic Neighbour Embedding

1. Summary of First Graduation Project

In the first graduation project I worked on MNIST and Fashion MNIST datasets. The MNIST and fashion MNIST data sets are great for starting image processing. In the MNIST, many operations such as training the machine, performance, accuracy, error analysis were carried out using the PyCharm editor. A classification study was carried out on the MNIST data set. Mainly focused on SGD classifier. This classifier has the advantage of being capable of handling very large dataset. Then performance measurements were made with different methods such as confusion matrix. Then, the subject of trade off with precision recall was discussed. Here, information was given about how the model should be preferred depending on the subject, whether it will have high sensitivity, low recall or low sensitivity, high recall. The lowest threshold that gives us at least 90% precision point has been selected. Additionally, the model's precision recall graph was drawn according to threshold and observations were made. Then ROC (receiver operating characteristic) curve tool used for classifier to observe the performance of the model. ROC plot consists of true positive rate (another name for recall) and false positive rate (FPR). It is similar to precision recall curve.

One way to compare classifiers is to measure the area under the curve (AUC). A perfect classifier will have a ROC AUC equal to 1. After the dataset trained with random forest classifier and compare to its ROC curve and ROC AUC score to those of the SGD classifier. And then he SGD classifier and random forest's ROC curve plotted and compared. Random forest is much better than SGD classifier.

After that multiclass classification used to perform classification with multiple binary classifiers. One way to create a system that can classify the digit images into 10 classes (from 0 to 9) is to train 10 binary classifiers, one for each digit (a 0-detector, a 1-detector, a 2- detector, and so on). Then, to classify an image, the decision score was taken from each classifier for that image and the class whose classifier gave the highest score was selected. This is called the one-versus-all (OvR) strategy (also called one-versus-all). Also, there is another strategy called OvO means one versus one strategy. It trains a binary classifier for every pair of digits: one to distinguish 0s and 1s, another to distinguish 0s and 2s, another for 1s and 2s, and so on. For most binary classification algorithms, OvR is preferred. It is related with speed of the model. For this data set sklearn automatically used OvO strategy (But we can choose any method.). However, training an SGDClassifier (or a RandomForestClassifier) is just as easy. Scikit-Learn

did not have to run OvR or OvO because SGD classifiers can directly classify instances into multiple classes.

Then error analysis performed with confusion matrix. After focused on multilabel classification. Such a classification system that outputs multiple binary tags is called a multilabel classification system. A multilabel array containing two target labels for each digit image: the first indicates whether or not the digit is large (7, 8, or 9), and the second indicates whether or not it is odd. And next create a K neighbors classifier instance (which supports multilabel classification, though not all classifiers do), and we train it using the multiple targets array. Then harmonic mean of precision recall F_1 score measured for each individual label, then simply compute the average score.

The last type of classification we will discuss is called multi-output-multi-class classification (or simply multi-output classification). This is a simple generalization of multi-label classification, where each label can be multi-class (i.e. have more than two possible values). Here, the noisy image filtering process is done. The classifier's output is multilabel (one label per pixel) and each label can have multiple values (pixel intensity ranges from 0 to 255). It is thus an example of a multioutput classification system.

In the second part of the graduation project fashion MNIST data set used for classification process with CNN (Convolutional Neural Network) algorithm which is one of the deep learning architectures. Fashion MNIST is a dataset consisting of 28x28 pixel grayscale images of 70,000 clothing items in 10 different categories. The data was downloaded with the tensor flow library, and the normalization process was performed, that is, the pixel data was reduced to be between 0 and 1. This process is carried out to increase the performance of the model. Then the CNN model was defined. The model has been compiled. Optimizer, loss, and metrics parameters were set to determine how to train the model. The optimizer is the algorithm used to update the weights of the model. Loss is a function that measures the accuracy of the model's predictions. Metrics are performance indicators that are tracked during training and testing. The model has been trained. The model was trained for 5 epochs on the training data. Epoch means that the entire data set is processed once by the model. As a result of the training, the accuracy and loss of the model on the training data were printed on the screen. The model was tested. The model was evaluated on the test data and its accuracy and loss on the test data were printed on the screen.

2. Introduction

In this graduation project, digital image processing will be examined with machine learning algorithms. Image processing used for a wide range of applications in environment, agriculture, military, industry, and medical science. In this graduation project, I worked on MNIST to displaying MNIST images in 2 dimensions with t-SNE and classifying MNIST images by reducing them to 2 dimensions with linear discriminant analysis were performed. In this project, I will also briefly talk about the previous graduation project.

3. MNIST

The MNIST database (Modified National Institute of Standards and Technology) is a large database of handwritten digits that is commonly used for training various image processing systems.

In this graduation project the MNIST dataset used, which is a set of 70,000 small images of digits handwritten by high school students and employees of the US Census Bureau. Each image is labeled with the digit it represents. This set has been studied so much that it is often called the "hello world" of Machine Learning: whenever people come up with a new classification algorithm, they are curious to see how it will perform on MNIST, and anyone who learns Machine Learning tackles this dataset sooner or later.

Datasets loaded by Scikit-Learn. There are 70,000 images, and each image has 784 features. This is because each image is 28 × 28 3pixels, and each feature simply represents one pixel's intensity, from 0 (white) to 255 (black). Let's choose one digit from the dataset. All we need to do is grab an instance's feature vector, reshape it to a 28 × 28 array, and display it using Matplotlib's imshow () function:

Figure 1. Some digit plotted from data.

This looks like a 3, its code shown in appendix section. Also, we can see in figure 2 shows a few more images from the MNIST dataset.

Figure 2. Digits from the MNIST dataset.

Test set should create and set it aside before inspecting the data closely. The MNIST dataset is already split into a training set (the first 60,000 images) and a test set (the last 10,000 images). The training set is already shuffled for us, which is good because this guarantees that all cross-validation folds will be similar (we don't want one-fold to be missing some digits).

4. T-SNE

T-SNE (t-distributed stochastic neighbor embedding) is a statistical method for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional map. It is based on Stochastic Neighbor Embedding originally developed by Geoffrey Hinton and Sam Roweis, where Laurens van der Maaten proposed the *t*-distributed variant.

Embedding means embedding the element in the d-dimensional space into 2 or 3 dimensions. Assign each point in the multi-dimensional space to corresponding point. Stochastic mean probabilistic, random. And t-distribution or student's t distribution mean a probabilistic distribution type sometimes called Cauchy distribution.

T-SNE is a nonlinear dimensionality reduction technique for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions. Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability.

The t-SNE algorithm consists of two main stages. First, t-SNE creates a probability distribution over pairs of high-dimensional objects such that similar objects are assigned a higher probability, while dissimilar points are assigned a lower probability. Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map and minimizes the Kullback-Leibler deviation (KL deviation) between the two distributions with respect to the locations of the points on the map.

T-SNE has been used for visualization in a wide range of applications, including genomics, science, natural language processing, music analysis, cancer research, biological research, geological domain interpretation, and biomedical signal processing.

T-SNE tries to cluster similar elements in dimension reduction.

5. Linear Discriminant Analysis

Linear discriminant analysis (LDA), normal discriminant analysis or discriminant function analysis is a generalization of Fisher's linear discriminant. This is a method used in statistics and other fields, to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification. Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is used for supervised classification problems in machine learning.

LDA reduces the dimensionality of data by finding linear functions that make the classes as distinct as possible. It looks for the linear functions that have the highest ratio of the variance between classes to the variance within classes. That means, it searches for the best ways to split the feature space into different class regions. LDA assumes that the data has a Gaussian distribution and that the covariance matrices of the different classes are equal. It also assumes that the data is linearly separable, meaning that a linear decision boundary can accurately classify the different classes.

LDA has some advantages and limitations. The advantages are it is simple and computational efficient algorithm, it can work well when the number of attributes much larger than number of training samples. But there are some disadvantages including it assumes that the covariance matrices of the different classes are equal which may not be true in some datasets. It assumes that the data is linearly separable.

In real world application it is used in medical, face recognition, customer identification, robotics and for some predictions.

6. Material and Methods

In this section, detailed information is given about how t-SNE and LDA algorithms work. An attempt has been made to explain the mathematics behind these algorithms.

6.1 Mathematical formulation of t-SNE

Before explaining t-SNE I explained the SNE which is head of t-SNE. SNE (Stochastic Neighbor Embedding) is convert all pairwise distances between high dimensional data to probability. Stochastic Neighbor Embedding (SNE) starts by converting the high-dimensional Euclidean distances between datapoints into conditional probabilities that represent similarities. The similarity of datapoint x_j to datapoint x_i is the conditional probability, $p_{j|i}$, that x_i would pick x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i . For nearby datapoints, $p_{j|i}$ is relatively high, whereas for widely separated datapoints, $p_{j|i}$ will be almost infinitesimal.

$$p_{j|i} = \frac{\exp\left(-\left\|x_i - x_j\right\|^2 / 2{\sigma_i}^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2{\sigma_i}^2\right)} \tag{1}$$

In the formula that above represents the conditional probability of $p_{j|i}$. Note that $p_{i|i} = 0$ because we are only interested in modeling pairwise similarities. And σ_i is variance of the Gaussian that is centered on datapoint x_i . Sigma in SNE depends on the point. So, we don't define unique variance over the whole space. The intuition is that density od points varies in different part of the space. $p_{j|i}$ is not symmetric means $p_{j|i} \neq p_{i|j}$ because we change the variance. This $p_{j|i}$ conditional probability is defined for high dimension (X-space).

For the low-dimensional (y space) counterparts y_i and y_j of the high-dimensional datapoints x_i and x_j , it is possible to compute a similar conditional probability, which we denote by $q_{j|i}$.

$$q_{j|i} = \frac{\exp\left(-\left\|y_i - y_j\right\|^2\right)}{\sum_{k \neq i} \exp\left(-\|y_i - y_k\|^2\right)} \tag{2}$$

Also because of only interested in modeling pairwise similarities, we set $q_{i|i} = 0$. In theory mapping points y_i and y_j correctly model the similarity between the high-dimensional datapoints x_i and x_j, the conditional probabilities $p_{j|i}$ and $q_{j|i}$will be equal. But in practical there are some mismatches. We assume $\sigma = \frac{1}{\sqrt{2\pi}}$ for whole space. Also, it doesn't matter what

we choose, it scales the space. A natural measure of the faithfulness with which $q_{j|i}$ models $p_{j|i}$ is the Kullback-Leibler divergence (which is in this case equal to the cross-entropy up to an additive constant). SNE minimizes the sum of Kullback-Leibler divergences over all datapoints using a gradient descent method. The cost function C is given down below.

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} log \frac{p_{j|i}}{q_{j|i}} \quad (3)$$

Our purpose is to minimize this function. P_i represents the conditional probability distribution over all other datapoints given datapoint x_i, and Q_i represents the conditional probability distribution over all other map points given map point y_i. Kullback-Leibler divergence is not symmetric so there are some different types of error in the pairwise distances in the low-dimensional map are not weighted equally. In particular, there is a large cost for using widely separated map points to represent nearby datapoints (i.e., for using small $q_{j|i}$ to model a large $p_{j|i}$), but there is only a small cost for using nearby map points to represent widely separated datapoints. This small cost comes from wasting some of the probability mass in the relevant Q distributions. In other words, the SNE cost function focuses on retaining the local structure of the data in the map (for reasonable values of the variance of the Gaussian in the high-dimensional space, σ_i). Also, small $p_{j|i}$ is modeled by large $q_{j|i}$ leads to low cost.

The loss is optimized with using a gradient descent algorithm. The gradient is shown in equation 4.

$$\frac{\partial C}{\partial y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j) \quad (4)$$

In order to speed up the optimization the gradient update with a momentum term is used and it is shown in equation 5.

$$\gamma^{(t)} = \gamma^{(t-1)} + \eta \frac{\partial C}{\partial \gamma} + \alpha(t)(\gamma^{(t-1)} - \gamma^{(t-2)} \quad (5)$$

$\gamma^{(t)}$ indicates the solution at iteration t, η indicates the learning rate, and α(t) represents the momentum at iteration t.

Any particular value of σ_i induces a probability distribution, P_i, over all of the other datapoints. This distribution has an entropy which increases as σ_i increases. SNE performs a binary search for the value of σ_i that produces a P_i with a fixed perplexity that is specified by the user.

The perplexity is defined as equation 6.

$$Perp(P_i) = 2^{H(P_i)} \tag{6}$$

$H(P_i)$ is the Shannon entropy of P_i measured in bits. And $H(P_i)$ is shown in equation 7.

$$H(P_i) = -\sum_j p_{j|i} log_2 p_{j|i} \tag{7}$$

The perplexity is number of neighbors. The performance of SNE is robust to changes in the perplexity, it depends on size of data. It is robust between 5 to 50 but it maybe 100.

Now let's come to t-SNE. Although SNE constructs reasonably good visualizations, it is hampered by a cost function that is difficult to optimize and crowding problem.

There are two different ways between SNE and t-SNE. First one is to compute cost they wanted to compute cost easier in t-SNE, so they made high dimensional pairwise probability $p_{j|i}$ symmetric. In SNE sigma is varies in different points it was not same for whole space, but in t-SNE it is constant for whole space.

The second is to use the student t-distribution instead of Gaussian to calculate the similarity between two points in low-dimensional space. t-SNE employs a heavy-tailed distribution in the low-dimensional space to reduce both the crowding problem and the optimization problems of SNE.

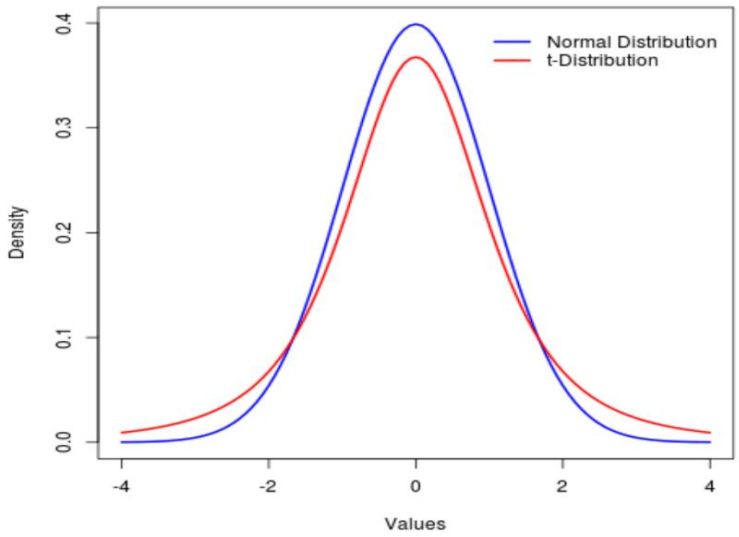

Figure 3. Normal distribution and t-distribution.

T-distribution has longer tails compared to Gaussian distribution. As shown in figure 3. It goes to zero more smoothly. So, we have more volume to accommodate points that are captured by that probability.

When mapping the distances between neighbors in d-dimensional space to 2D or 3D dimensions, sometimes it is impossible to preserve distances in all neighbors this is called crowding problem. The points getting crowded to impossible preserve neighbor distances in low dimensions. So, t-distribution used to resolve the crowding problem. So, the best trade off trying to catch.

Sigma is depending on the initial point. It is constant for whole space. The pairwise similarities in the high-dimensional space p_{ij} is shown below.

$$p_{ij} = \frac{\exp\left(-\left\|x_i - x_j\right\|^2 / 2\sigma^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma^2\right)} \tag{8}$$

But there are some problems when a high-dimensional datapoint x_i is an outlier (i.e., all pairwise distances $\left\|x_i - x_j\right\|^2$ are large for x_i). For such an outlier, the values of p_{ij} are extremely small for all j, so the location of its low-dimensional map point y_i has very little effect on the cost function. As a result, the position of the map point is not well determined by the positions of the other map points. They solve this problem by defining the joint probabilities p_{ij} in the high-dimensional space to be the symmetrized conditional probabilities, that is, they

set $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2n}$. This ensures that $\sum_j p_{ij} > \frac{1}{2n}$ for all datapoints x_i. As a result, each datapoint x_i makes a significant contribution to the cost function.

In the low dimensional pairwise similarities q_{ij} is become like equation 9.

$$q_{ij} = \frac{\frac{1}{1+\|y_i - y_j\|^2}}{\sum_{k \neq i} \frac{1}{1+\|y_i - y_k\|^2}} \tag{9}$$

They are symmetric so, $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$ for ∀i, j. The cost function is again KL divergence. Our goal is to make cost minimum.

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{10}$$

For optimize the cost function the gradient descent algorithm used as same as SNE but little different and it is shown in equation 11.

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)\,(1 + \|y_i - y_j\|^2)^{-1} \tag{11}$$

The algorithm of t-SNE.

Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

Data: data set $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$,
cost function parameters: perplexity *Perp*,
optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, \ldots, y_n\}$.
begin
  compute pairwise affinities $p_{j|i}$ with perplexity *Perp* (using Equation 1)
  set $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2n}$
  sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, \ldots, y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
  for $t=1$ **to** T **do**
    compute low-dimensional affinities q_{ij} (using Equation 4)
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$
  end
end

6.2 Mathematical Formulation of LDA

Before we get into the details of LDA, let's write Naive Bayes classification algorithm, which forms the basis for LDA. Naive Bayes is derived from Bayes rule shown in equation 12.

$$P(C_k|X) = \frac{P(X|C_k)P(C_k)}{P(X)} \tag{12}$$

$P(C_k|X)$ is posterior probability, $P(X|C_k)$ is likelihood $P(C_k)$ the prior probability of class and the $P(X)$ is prior probability of predictor. X is feature vector and C_k is class variable. Using the Naive independence assumption, which states that,

$$P(X|C_k) = P(X = x_1, \dots x_n|C_k) \prod_{i=1}^{n} P(x_i|C_k) \tag{13}$$

Then the posterior probability can be written as,

$$P(X|C_k) = \frac{P(C_k) \prod_{i=1}^{n} P(x_i|C_k)}{P(X)} \tag{14}$$

The Naive Bayes classification problem then becomes for different class values of C_k find the maximum of $P(C_k) \prod_{i=1}^{n} P(x_i|C_k)$ and this can be formulated as,

$$\hat{C} = argmax_{Ck}\, P(C_k) \prod_{i=1}^{n} P(x_i|C_k) \tag{15}$$

Discriminant analysis can be viewed as a 5-step procedure:

1) Calculate prior probabilities. The prior probability of class $P(C_k)$ could be calculated as the relative frequency of class C_k in the training data.
2) Test of variances homogeneity. Use Bartlett's test to test if K samples are from populations(classes) with equal variance-covariance matrices. The result of this test will determine whether to use linear discriminant analysis.
   For LDA accept variance-covariance matrices as $\sum_1 = \sum_2 = \cdots = \sum_{K-1}$.
3) Estimate parameters of the likelihoods**.** Estimation of the parameters (e.g. $\mu_i \; and \; \sum$) of the conditional probability density functions $P(X|C_k)$ from the training data. Here, the standard assumption that the data are multivariate normally (Gaussian) distributed.
4) Compute discriminant functions**.** This is the rule to classify the new object into one of the known populations.

5) Estimate classification performance.

LDA is done with density estimation which is one of the classification approaches.

Assume that in population π_i the probability density function of x is multivariate normal (Gaussian) with mean vector μ_i and variance-covariance matrix Σ same for all populations. And d is the number of features. Then the formula for this normal (Gaussian) probability density function is shown in equation down below.

$$P(X|\pi_i) = \frac{1}{(2\pi)^{d/2}|\sum|^{1/2}} \exp\left(-\frac{1}{2}(X-\mu_i)^T\sum^{-1}(X-\mu_i)\right) \quad (16)$$

According to the Naive Bayes classification algorithm. We know that we classify the example to the population for which $P(\pi_i)\ P(X|\pi_i)$ is the maximum. For the ease of calculation, we also take a log transform.

LDA is used when the variance-covariance matrices for all populations are homogeneous. In LDA, our decision rule is based on the Linear Score Function, a function of the population means for each of our g populations, μ_i , and the pooled variance-covariance matrix. The linear score function is shown in equation 17.

$$\begin{aligned} s_i^L(X) &= -\frac{1}{2}{\mu_i}^T\sum^{-1}\mu_i + {\mu_i}^T\sum^{-1}X + logP(\pi_i) = \\ &= d_{i0} + \sum_{j=1}^{p} d_{ij}x_j + logP(\pi_i) = \\ &= d_i^L(X) + logP(\pi_i) \end{aligned} \quad (17)$$

$d_i^L(X)$ is refers to linear and the other is constant. $d_{i0} = -\frac{1}{2}{\mu_i}^T\sum^{-1}\mu_i$ and $d_{ij} = jth$ element of ${\mu_i}^T\sum^{-1}$. Assume we have feature vector $X = (x_1, x_2, \dots, x_n)$ the linear score function is computed for each population, and we classify the example into the population which has the largest linear score function. This is equivalent to classifying to the population for which the posterior probability of membership is largest.

In practice we don't know the parameters of Gaussian and we need to estimate them using our training data. These are π_i, μ_i and $\sum$.

Estimating the prior π_i is the easiest one. The formula is show below.

$$\pi_k = \hat{P}(y = k) = \frac{n_k}{n} \tag{18}$$

It is simple assume we have two class. There are 10 instances from class 1 and if we calculate prior probability, it is 0.1 and for class 2 it is 0.9.

For class or population means we do the calculation that shown below.

$$\mu_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i \tag{19}$$

For the variance-covariance matrix first we calculate each variance-covariance of each class then take the average of them and choose it for common variance-covariance matrix.

$$\hat{\sum}_k = \frac{1}{n_k - k} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T \tag{20}$$

The equation that shown above is variance-covariance matrix for each class. In LDA we must calculate a common variance-covariance matrix, so we take an average of all the variance-covariance matrices that shown in equation 21. n_r is total element of each class.

$$\sum = \frac{\sum_{r=1}^{k} (n_r \sum_r)}{\sum_{r=1}^{k} (n_r)} \tag{21}$$

n_r is number of data points in class r $\sum_r$ is covariance of class r and n is the total number of data points and k is the number of classes.

Sometimes score function shown with symbol δ. The decision boundary is the set of points for which classes are equally probable means that $\delta_k(x) = \delta_l(x)$. Then the decision boundaries defined as in equation 22.

$$-\frac{1}{2}\mu_k{}^T \sum^{-1} \mu_k + \mu_k{}^T \sum^{-1} X + logP(\pi_k) = -\frac{1}{2}\mu_l{}^T \sum^{-1} \mu_l + \mu_l{}^T \sum^{-1} X + logP(\pi_l) \tag{22}$$

6.2.1 Dimensionality Reduction

LDA projects data from a D dimensional feature space down to a D' (D>D') dimensional space in a way to maximize the variability between the classes and reducing the variability within the classes. LDA try to find a new axis to separate the data as possible. Let's understand the behind of dimensionality reduction with 2 classes and d dimensions.

Let's understand some parameters. W (weights or eigenvector) be a unit vector onto which the data points are to be projected (took unit vector as we are only concerned with the direction). N is number of instances ($N = N_1 + N_2$). Means of classes before projection m_i (μ_i). Means of classes after projection is $M_i = W^T m_i$.

X which the samples on the feature space and $W^T X$ denotes the data points after projection. As we can see in the figure below.

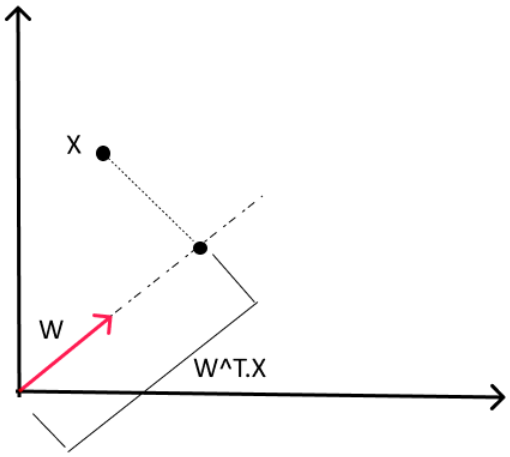

Figure 4. Projection.

The between class scatter denotes as S_B and it measures the distance between class means. And within class scatter denotes as S_w and it measures the spread around means of each class.

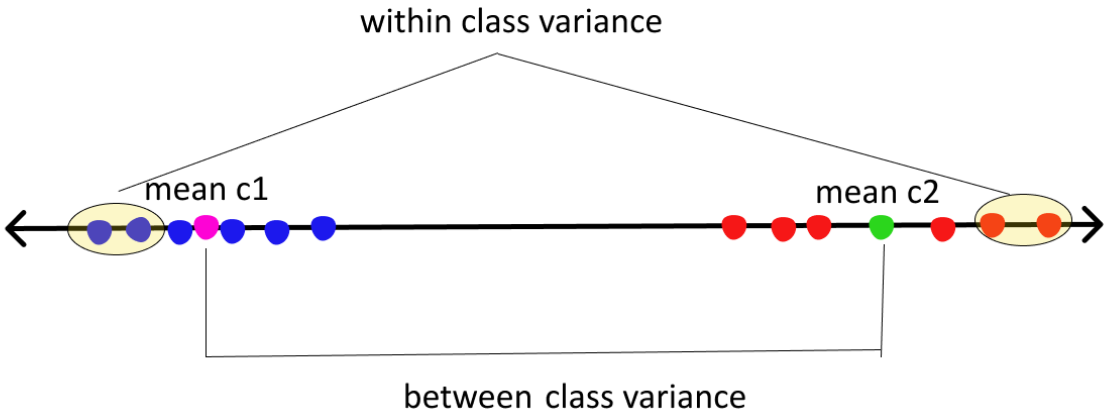

Figure 5. Projected data in a line.

The means and between class distance shown in the figure above.

As Fisher's LDA the cost function is shown in the figure 5.

$$J(W) = \frac{(M_1 - M_2)^2}{{S_1}^2 + {S_2}^2} \tag{23}$$

The numerator here is between class scatter while the denominator is within-class scatter. We want to maximize the cost function for better separation. We want to minimize the variation(scatter) within categories (denominator). And we want to maximize the distance between means. To maximize the above function, we need to first express the above equation in terms of W, and it is shown below.

$$J(W) = \frac{W^T S_B W}{W^T S_w W} \tag{24}$$

LDA can be generalized for multiple classes. Here are the generalized forms of between-class (S_B) and within-class matrices S_w.

$$S_B = \sum_{i=1}^{c} N_i (m_i - m)(m_i - m)^T \tag{25}$$

$$m = \frac{1}{N} \sum_{i=1}^{n} x_i \tag{26}$$

$$m_i = \frac{1}{N_i} \sum_{x(n) \in C_i} x(n) \tag{27}$$

The m is mean of all data points and m_i is mean of specific class. Within scatter is shown in equation 28.

$$S_w = \sum_{k=1}^{c} S_k \tag{28}$$

$$S_k = \sum_{x(n) \in C_i} (x(n) - m_i)\,(x(n) - m_i)^T \tag{29}$$

C is number of classes.

$$W = eig(S_w^{-1} S_B) \tag{30}$$

The top equation gives us the directions with maximum separation.

Note: S_B is the sum of C different rank 1 matrices. So, the rank of $S_B \leq C - 1$. That means we can only have C-1 eigenvectors. Thus, we can project data points to a subspace of dimensions at most C-1. The equation 25 gives between-class scatter. Finally, eigen decomposition of $(S_w^{-1}S_B)$ gives the desired eigenvectors. And the equation 29 gives scatter for each of classes and equation 28 adds all of them to give within-class scatter.

7. Implementation of t-SNE to MNIST Data Set

For implementation of t-SNE to MNIST data set I used python programming language. For t-SNE and MNIST data set used scikit learn library. The data loaded and then split to feature data and target data. After the feature data (X) normalized. Then t-SNE model performed with some parameters like n_component, perplexity, and iteration. Different perplexity and iteration applied to obtain better embedded visuals. For visualization I used matplotlib and seaborn libraries. You can access the code in Appendix section. Let's see the plots with different iteration and perplexity. By the way perplexity is number of neighbors.

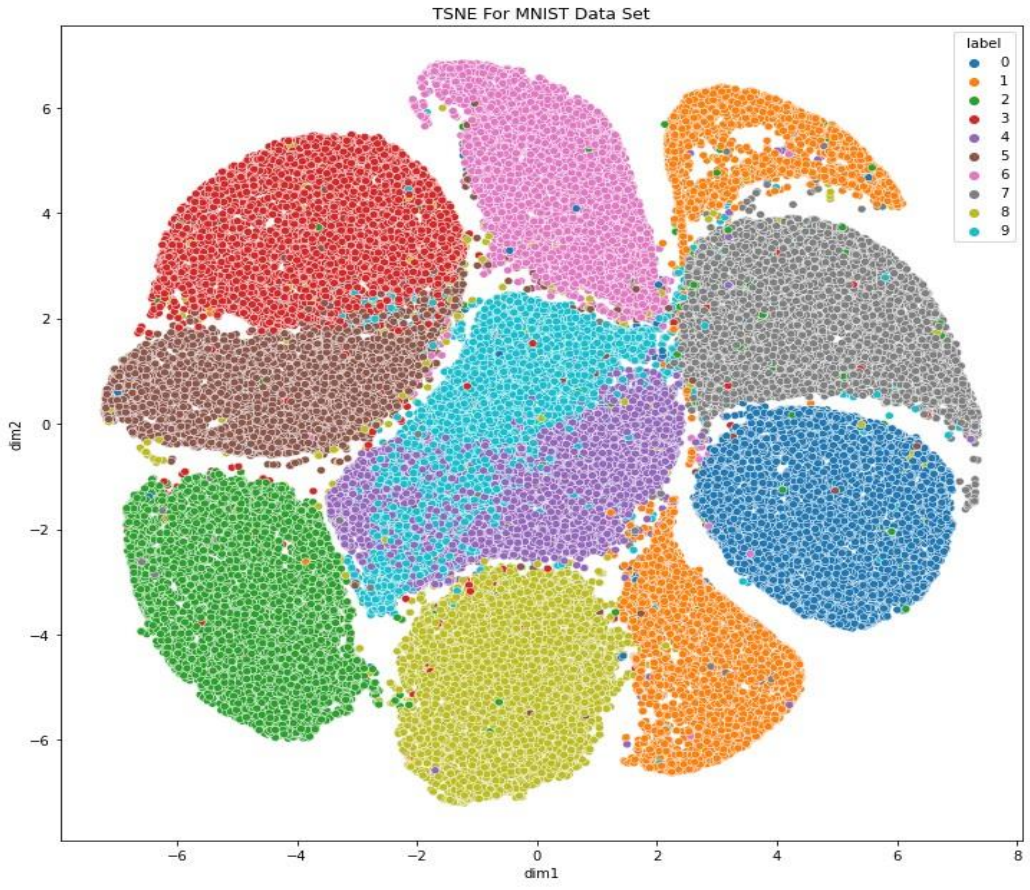

Figure 6. TSNE plot 1.

In the figure 6 we can see the plot with perplexity 3 and iteration 300. The figure above the target values are not separated well so we have to try different perplexity and iterations for better plot.

After a few tries I achieved better plot to visualize the MNIST data set. In the figure below we can see the plot that has perplexity 30 and iteration 1500.

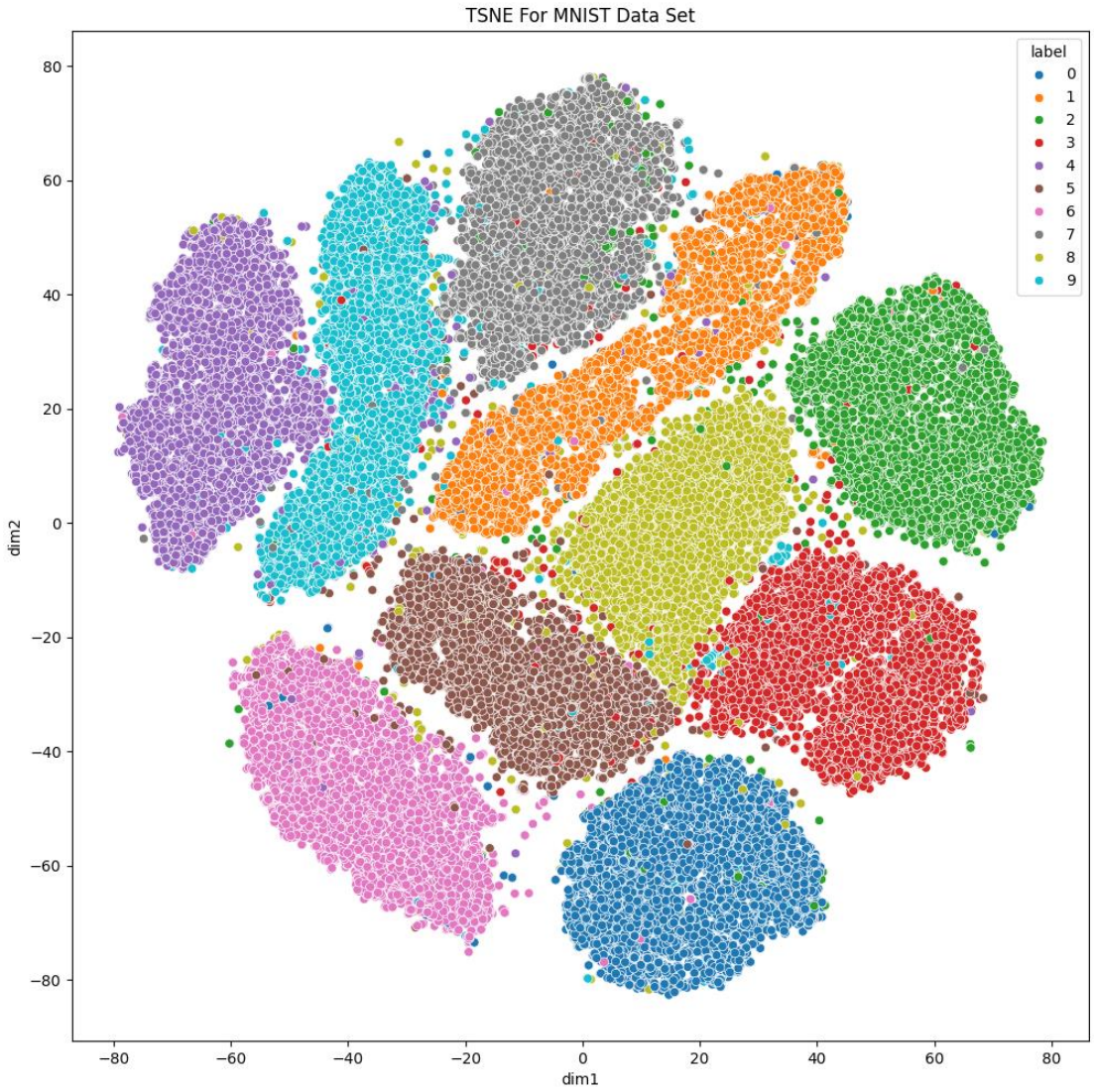

Figure 7. TSNE plot 2.

As a result, I used t-SNE algorithm to visualize high dimension MNIST data set in low dimension (2D).

8. Implementation of LDA to MNIST Data Set

For implementation of LDA to MNIST data set I used python programming language. For LDA and MNIST data set used scikit learn library. Then I split the data for train and test data. Then I created LDA model and fitted my train data to model. The model default reduces the dimension to 2 dimensions. After I applied the same operation to test data, reduced the dimension. Then I get the prediction of test data and compared with actual test data result. After that I get 87.30 accuracy. It is good classification result. You can access the code from appendix section.

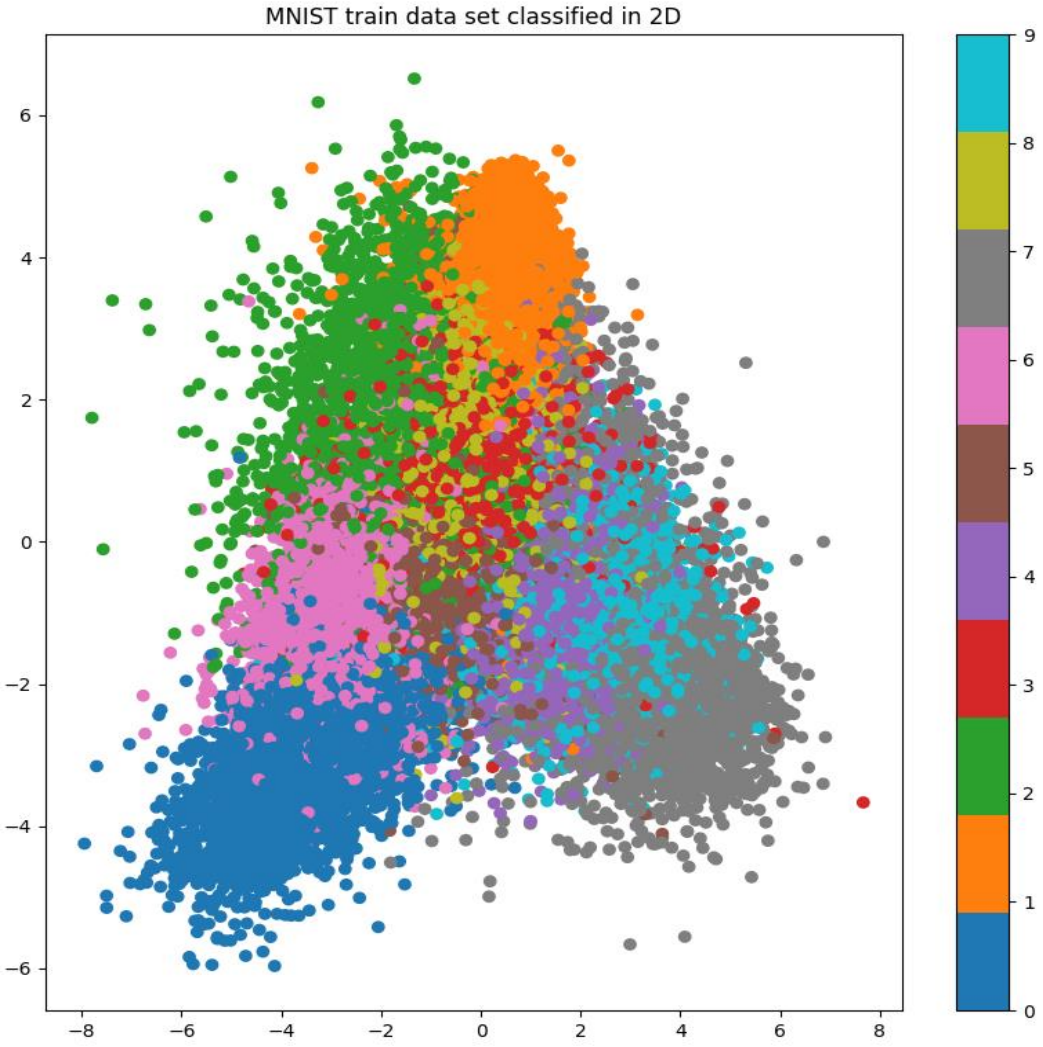

Figure 8. LDA train data

In the figure 8 we can see the train data of MNIST data that reduced to 2D.

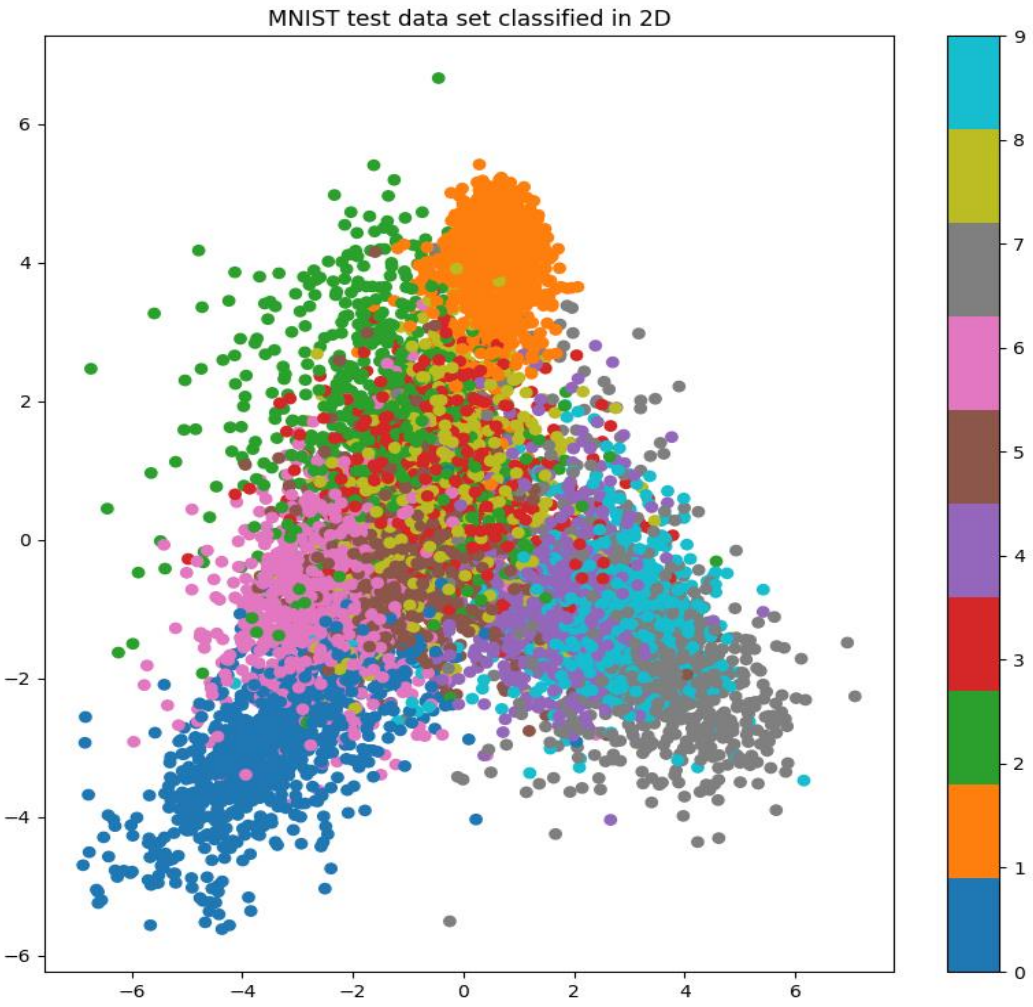

Figure 9. LDA test data

In the figure 9 we can see the test data of MNIST data that reduced to 2D.

Conclusion

In conclusion in this graduation project, I first gave information from my first graduation project, like a brief summary. Then, information was given about the MNIST data set to be worked on. Afterwards, information was given about t-SNE and LDA, which are the two subjects of this graduation project, and about its mathematical background. Then, the multidimensional MNIST data set was reduced to two dimensions and visualized with t-SNE using the Python programming language. After that, the other topic, linear discriminant analysis examined. The MNIST data set was reduced to two dimensions and visualization was performed with LDA. And then classification was performed with the test data set and the accuracy value was printed.

Appendix

```
# For plotting digit and some digits of MNIST data set

import sklearn
assert sklearn.__version__ >= "0.20"
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
mnist.keys()
X, y = mnist["data"], mnist["target"]
some_digit = X[12]
some_digit_image = some_digit.reshape(28, 28)
plt.imshow(some_digit_image, cmap=mpl.cm.binary)
plt.axis("off")

plt.show()

def plot_digits(instances, images_per_row=10, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    n_rows = (len(instances) - 1) // images_per_row + 1
    n_empty = n_rows * images_per_row - len(instances)
    padded_instances = np.concatenate([instances, np.zeros((n_empty, size * size))], axis=0)
    image_grid = padded_instances.reshape((n_rows, images_per_row, size, size))

    big_image = image_grid.transpose(0, 2, 1, 3).reshape(n_rows * size,
                                                         images_per_row * size)
    plt.imshow(big_image, cmap = mpl.cm.binary, **options)
    plt.axis("off")

plt.figure(figsize=(9,9))
example_images = X[300:400]
plot_digits(example_images, images_per_row=10)
plt.show()
```

```
# Code for TSNE

# Import libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.manifold import TSNE
import seaborn as sns

# Load the MNIST 784 data set
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
X = mnist.data / 255.0  # Normalize the pixel values
y = mnist.target  # Get the labels

# Perform TSNE with 2 components
tsne_model = TSNE(n_components=2, perplexity=30, n_iter=1500)
tsne_results = tsne_model.fit_transform(X)

# Convert the results to a pandas data frame
feat_cols = ['dim1', 'dim2']
tsne_df = pd.DataFrame(tsne_results, columns=feat_cols)
tsne_df['label'] = y

# Plot the TSNE results with labels
plt.figure(figsize=(12, 12))
sns.scatterplot(
    data=tsne_df, x="dim1", y="dim2",

    palette=sns.color_palette("tab10", 10),  # tab10  is color type
    hue="label",

    legend="full",
    hue_order=sorted(tsne_df['label'].unique())

)
plt.title("TSNE For MNIST Data Set")
plt.show()
```

Code for LDA

```
from sklearn.datasets import fetch_openml
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score

mnist = fetch_openml('mnist_784', version=1, as_frame=False)
X = mnist["data"]
y = mnist["target"]
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

lda = LinearDiscriminantAnalysis(n_components=9) # ncomponent comes from label of data
X_train_lda=lda.fit_transform(X_train, y_train)

test_lda = lda.transform(X_test)
test_predict = lda.predict(X_test)

plt.figure(figsize=(10, 10))
plt.scatter(
   X_train_lda[:,0],
   X_train_lda[:,1],
   c=y_train.astype(int),
   cmap='tab10'
   )
plt.colorbar()
plt.title('MNIST train data set classified in 2D')
plt.show()

plt.figure(figsize=(10, 10))
plt.scatter(
   test_lda[:,0],
   test_lda[:,1],
   c=y_test.astype(int),
   cmap='tab10'
   )
plt.colorbar()
plt.title('MNIST test data set classified in 2D')
plt.show()
acc = accuracy_score(y_test, test_predict)
print(f'Accuracy: {acc:.4f}')
```

References

Books

Hands On Machine Learning with Scikit Learn Keras and TensorFlow 2nd Edition Aurelien Geron

Websites

1) https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding
2) https://lvdmaaten.github.io/tsne/
3) https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf
4) https://opentsne.readthedocs.io/en/latest/tsne_algorithm.html
5) https://shuzhanfan.github.io/2018/07/understanding-mathematics-behind-lda/
6) https://web.stanford.edu/~lmackey/stats202/content/lec9-condensed.pdf
7) https://www.analyticsvidhya.com/blog/2021/08/a-brief-introduction-to-linear-discriminant-analysis/
8) https://www.statology.org/t-distribution-python/
9) https://www.youtube.com/watch?v=4GBgqmq0XAY
10) https://www.youtube.com/watch?v=azXCzI57Yfc&t=730s
11) https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
12) https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html