HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

MASTER THESIS

Mr.
Rohit Mallinath Shetgar

Comparing visualizations of dimensionality reduction methods Autoencoders and t-SNE

2020

Faculty of **Applied Computer and Life Sciences**

MASTER THESIS

Comparing visualizations of dimensionality reduction methods Autoencoders and t-SNE

Author:
Rohit Mallinath Shetgar

Study Programme:
Applied Mathematics for Digital Media

Seminar Group:
MA16W1-M

First Referee:
Prof. Dr. Thomas Villmann

Second Referee:
Dr Marika Kaden

Mittweida, August 2020

Thanks to

my parents,
for filling confidence in me and blessing me.

Special thanks to

Prof. Dr. Thomas Villmann,
for the kind support and guidance,
Dr. Marika Kaden,
for proofreading the manuscript,
and giving valuable feedback
throughout the thesis.

Bibliographic Information

Shetgar, Rohit Mallinath: Comparing visualizations of dimensionality reduction methods Autoencoders and t-SNE, 41 pages, 20 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer and Life Sciences

Master Thesis, 2020

Abstract

We present dimensionality reduction methods like autoencoders and t-SNE for visualization of high-dimensional data into a two-dimensional map. In this thesis, we initially implement basic and deep autoencoders using breast cancer and mushroom datasets. Next, we build another dimensionality reduction method t-SNE using the same datasets. The obtained visualization results of the datasets using the dimensionality reduction methods are documented in the experiments section of the thesis. The evaluation of classification and clustering for the dimensionality reduction techniques is also performed. The visualization and evaluation results of t-SNE are significantly better than the other dimensionality reduction techniques.

I. Contents

II. List of Figures

III. List of Tables

1 Introduction

One of the challenges in handling big data is in creating effective human-computer interaction tools for rapidly customizable visual reasoning with respect to different applications. The high-dimensional data can be visually inspected by humans through an intuitive interface to identify the structural elements of the data like clusters, homogeneous regions, or outliers and depending on the cognitive capabilities of humans for speedy visual attention of structures and grouping of items [18].
The common problems in data mining deal with data with a large number of measurements or dimensions. The reduction of dimensionality makes it possible to extract knowledge from the data through visualization and to design and make use of effective classification schemes. The most important dimensions which hold most information for the task are kept as it is by performing dimensionality reduction and projecting the reduced dimensions onto others. The above steps help us significantly to visualize the data especially by mapping them in two or three dimensions which helps us in refraining from dealing with high dimensional data, instead we scan their low dimensional "summaries". The challenge for visualization is to insert a set of observations into a Euclidean feature-space, which preserves as nearly as possible their intrinsic metric structure [17].
Dimensionality reduction algorithms help in projecting high dimensional data to a low dimension and retaining important features as likely and reducing the irrelevant or repeated information.
Dimensionality reduction is an important part of machine learning to resolve expensive computational problems that involve images, videos, speech, and text. Machine learning involves two major branches supervised learning and unsupervised learning. In supervised learning, the artificial intelligence agent will have access to labels that are used in improving the performance on different tasks. In unsupervised learning, we do not have access to labels and the task for AI agent is not properly defined and the performance cannot be clearly measured. We work closely into unsupervised learning and get deep into different dimensionality reduction algorithms. Basically there are two branches of dimensionality reduction, first is known as a linear projection which deals in linearly projecting data from high-dimensional space to a low-dimensional space. The linear projection involves methods like principal component analysis, singular value decomposition, and random projection. The second is known as non-linear dimensionality reduction which involves multidimensional scaling, locally linear embedding, t-distributed stochastic neighbor embedding (t-SNE) and autoencoders are used with different types of autoencoders like deep autoencoders and basic autoencoders [9].

1.1 Overview of visualization methods

This section is based on the book from I. Goodfellow et al [20]. The performance of simple machine learning algorithms depends on how the data is represented and given to them. Suppose a machine learning algorithm is used to recommend patient for cesarean delivery, the artificial intelligence system cannot test the patient directly. Alternatively, the doctor gives the relevant information, like the existence or absence of a uterine scar. Every information collected by doctor in incorporating the representation of the patient is known as feature. The machine learning algorithms learns how to correlate every features of the patient with their associative outcomes.
The right features for the given task can solve the artificial intelligence systems particular task easily, subsequently these features can be given to the machine learning algorithms. However, it is difficult to know what features to be extracted for different task. For example, we would like to detect the car in photographs and as a feature to identity car we might use car wheels as a feature. The difficulty is describing about the car wheels in terms of pixel values. As the image can be affected by shadows and sun glaring falling on the metal parts of the wheel. To solve this difficulty, we need to use machine learning to identify not only the mapping of the representation to the output but the representation itself. This is known as representation learning.
The learnt representation quite often results in good performance than with hand designed representations. This leads to less human intervention and allows artificial intelligence systems to adapt for newer tasks. The standard example of the representation learning is autoencoders.
An autoencoder consists of an encoder function that transforms the input data into a different representation and the decoder function converts the new representation back into the original form. Autoencoders are trained to preserve as much information as possible when an input is run through the encoder and then the decoder, but are also trained to make the new representation have various nice properties. Different kinds of autoencoders aim to achieve different kinds of properties. When an input is passed through the encoder and then the decoder, the autoencoders are trained to retain most of the possible information.
The main difficulty in many artificial intelligence systems application in real-world is that there is a factor variation which affects every piece of data we observe. For example, the individual pixel of the red car can be very similar to black at night. So it might be very difficult to extract the abstract features from the raw data. So Deep learning solves this problem in representation learning by initiating representations that can be expressed in terms of other, simpler representations. The classic example of a deep learning model is the feed-forward deep network or multilayer perceptron (MLP).

1.1.1 Autoencoders

MLP is a mathematical function which maps some set of input values to output values. Many simpler functions are composed to form the function. Every application of the different mathematical function provides a new form of the input [20].
An autoencoder is considered as a special case of MLP as shown in Figure 1, which regresses the input data in its output layer. To do this, autoencoder does not require labeled data. Hence making it unsupervised, and also used to learn non-linear features from the input data [6].
It considers an encoder stage, a code layer, and a decoder stage. The input layer is represented as visible layer because it contains the data variables that we are able to see. The encoder stage consists of one or more hidden layers where it maps the input data to the code layer, which usually contains fewer neurons than the input layer [6].
A series of hidden layers used extracts the abstract features from the image (input). These layers are termed as “hidden” as their values are not given in the data; rather the model must determine which concepts are essential for analysing the relationships in the observed data.
The code layer captures all the important information which is needed to reconstruct the input in the output layer, through the decoder stage. This helps in low dimensional feature representation of the higher dimensional input data. Typically, the encoder and decoder are symmetric.
An autoencoder combined with MLP forms a network of trainable weights. The weights and the loss function is optimized by using stochastic gradient descent process [6].
An autoencoder with MLP, uses a backpropagation algorithm. It consists of two phases. In the first phase, error based on the reconstructed output and target output is calculated (forward phase) corresponding to the given input. In the second phase, the resultant error is back propagated to the network, based on this weight of the network are updated, usually until it converges [6]. Finally, the autoencoder with hyperspectral data as an input is shown in Figure 1, where the intensity at every wavelength corresponds to the value of each neuron in the input layer [6].

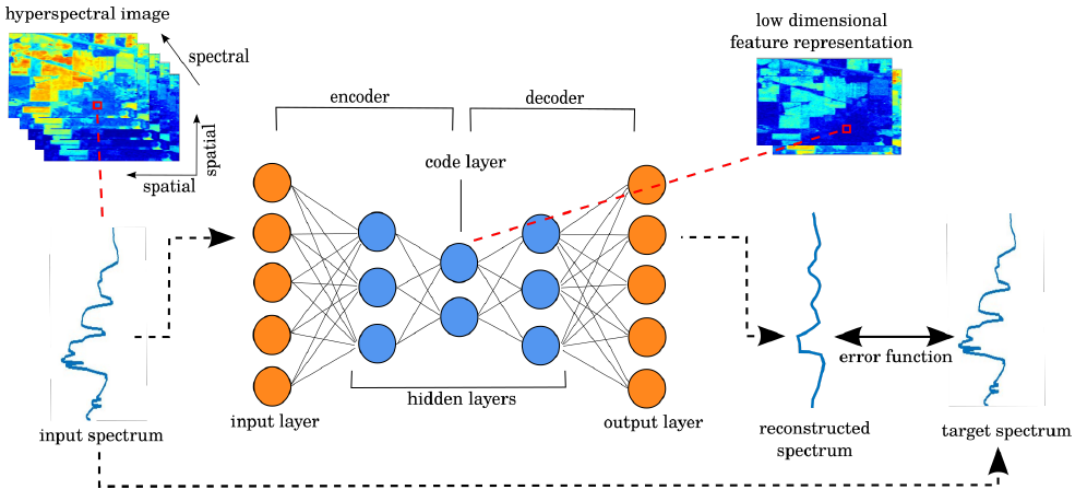

Figure 1.1: Autoencoders architecture [6]

1.1.2 Basic autoencoders

The basic autoencoder as shown in Figure 2, has only one hidden layer h and which sets target values to the given input x [21].
We consider an input $x \in R^n$, the hidden representation $h(x) \in R^m$ is given by

$$h(x) = f(W_1 x + b_1). \tag{1.1}$$

where $f(z)$ is a non-linear activation function, which is usually a logistic sigmoid function $f(z) = 1/(1+exp(-z))$ applied component-wise, $W_1 \in m \times n$ is a weight matrix and $b_1 \in R^m$ is a bias vector.

The output from the network as shown in Figure 2, maps the hidden representation h back to a reconstruction $\hat{x} \in R^n$

$$\hat{x} = f(W_2 h(x) + b_2) \tag{1.2}$$

where $W_2 \in n \times m$ is a weight matrix and $b_2 \in R^n$ is a bias vector.
We consider a set of input examples X, training autoencoder contains in finding parameters $\theta = \{W_1, W_2, b_1, b_2\}$ which minimizes the reconstruction error with respect to the objective function:

$$\mathscr{J}(\theta) = \sum_{x \in X} ||x - \hat{x}||^2 \tag{1.3}$$

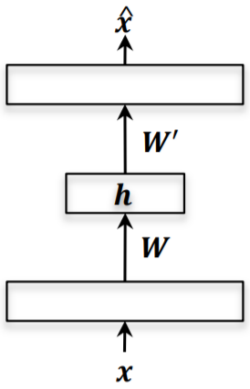

Figure 1.2: Basic autoencoder

The minimization can be realized by stochastic gradient descent that is used in training deep neural networks or MLP [21].

1.1.3 Deep autoencoders

A deep autoencoder is a variant of basic autoencoders but a feed-forward multilayer neural network where the desired output is input [5].
An autoencoder with more than one hidden layer is called a deep autoencoder and each additional hidden layer requires a pair of encoder and decoder as shown in Figure 3. The additional hidden layers can help in extracting more abstract features and better reconstruction of the input with minimum loss [5].
The code layer helps in capturing important features needed to reconstruct the input and also helps in low-dimensional representation of the input data.

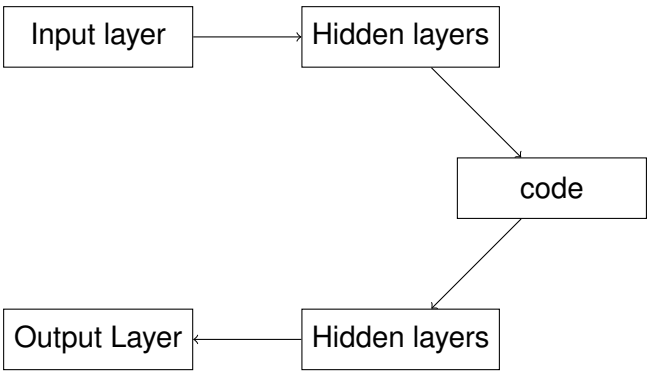

Figure 1.3: Deep autoencoder

1.1.4 Parameters used in autoencoders

During the construction of autoencoders each layer of the autoencoder is specified with a variety of parameters based on its activation function. Relu and sigmoid activation functions can be used for the encoder and decoder layers of the autoencoders [10]. The cost functions like mean squared error (MSE) and binary cross entropy are used to compile the autoencoder [10]. The most important hyperparameter of autoencoder is code size that is used for low-dimensional visualization of the high dimensional input data.

1.2 t-Distributed Stochastic Neighbor Embedding(t-SNE)

This section is based on the journal from L. Maaten and G Hilton [8]. We initially outline about stochastic neighbor embedding (SNE) published by Hilton and Roweis (2002), that forms the basis for t-Distributed Stochastic Neighbor Embedding (t-SNE).
SNE begins with the conversion of high-dimensional Euclidean distances between datapoints into conditional probabilities that represent similarities.
Alternatively, SNE can be applied to data sets that contains pairwise similarities between objects instead of high-dimensional vector representations of every object, given these similarities can be interpreted as conditional probabilities. The similarity between datapoint x_j and datapoint x_i is given by conditional probability $p_{j|i}$, that x_i would pick x_j as its neighbor provided neighbors were picked in proportion to their probability density under Gaussian centered (normally distributed) at x_i.
The probability $p_{j|i}$ for nearby datapoints drawn under this Gaussian will be relatively high, whereas for far away datapoints, $p_{j|i}$ will be extremely small or approaching zero.
The conditional probablity $p_{j|i}$ can be expressed mathematically as:

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k\neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)}, \tag{1.4}$$

where σ_i is known as variance of the Gaussian centered on datapoint x_i. The method of finding the value of σ_i is determined later in this section. The main focus is in modeling pairwise similarities, so the probability $p_{i|i}$ to zero. It is possible to calculate the similar conditional probability denoted as $q_{j|i}$ for low-dimensional datapoints y_i and y_j. The variance of the Gaussian that is employed to calculate the conditional probabilities are $q_{j|i}$ to $\frac{1}{\sqrt{2}}$.
Thus we model the similarity of map data points y_j and y_i by

$$q_{j|i} = \frac{exp(-||y_i - y_j||^2/2\sigma_i^2)}{\sum_{k\neq i} exp(-||y_i - y_k||^2/2\sigma_i^2)} \tag{1.5}$$

As we are interested in modeling pairwise similarities, $q_{i|i}$ is set to zero.
If the similarities are correctly modeled between map points y_i and y_j and high dimensional points x_i and x_j, the conditional probabilities $p_{j|i}$, $q_{j|i}$ will be equal. With this observation, SNE finds the low-dimensional data representation that minimizes the mismatch between $p_{j|i}$ and $q_{j|i}$.
The measure of trueness with which $q_{j|i}$ models $p_{j|i}$ is the Kullback-Leibler divergence. Gradient descent method is used by SNE to minimize the sum of Kullback-Leibler divergences over all datapoints.
The cost function is given by

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} log \frac{p_{j|i}}{q_{j|i}}, \tag{1.6}$$

where P_i stands for the conditional probability distribution over all other datapoints given datapoints x_i, and Q_i stands for the conditional probability distribution over all other maps given map point y_i. In particular, there is a high cost for using far away map points to represent nearby datapoints (that is for using a small $q_{j|i}$ to model a large $p_{j|i}$). But there is a low cost for using nearby map points to represent far away datapoints. In simple words, the SNE cost function tries to preserve the local structure of the data (high-dimension) in the map (low-dimension).
The important parameter to be chosen is variance σ_i of the Gaussian that is centered over each high-dimensional datapoint, x_i. There is no single value of σ_i that is optimal for all datapoints because density of data is likely to vary in different datasets. The probability distribution P_i is induced by any particular value of σ_i, over all other datapoints. This distribution is associated with entropy which increases as σ_i increases.
SNE operates binary search for the value of σ_i that produces P_i with a fixed perplexity given by the user. The perplexity is defined as

$$Perp(P_i) = 2^{H(P_i)}, \tag{1.7}$$

where $H(P_i)$ is the Shannon entropy of P_i measured in bits

$$H(P_i) = -\sum_j p_{j|i} \ log_2 \, p_{j|i}. \tag{1.8}$$

The perplexity can be interpreted as a smooth measure of effective number of neighbors. The performance of SNE is fairly powerful to the changes in the values of the perplexities. Typical values of perplexities are between 5 and 50.
The cost function as mentioned in Equation 1.6, can be minimized using a gradient

descent method. The gradient has a simple form as shown below :

$$\frac{\delta C}{\delta y_i} = 2\sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j). \tag{1.9}$$

The gradient may be explained as the resultant force induced by a set of springs between the map point y_i and all other map points y_j. The total force exerted by the spring is along the direction of $(y_i - y_j)$. There is a repulsion or attraction from the spring based on the distance between y_i and y_j in the map is too small or too big to represent the similarities between the two high-dimensional datapoints. The spring exerts a force between y_i and y_j which is proportional to its length and also its stiffness, which is mismatch $(p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})$ between the pairwise similarities of the map points and data points.
In order to initialize the gradient descent, the map points are sampled randomly from an isotropic Gaussian with small variance that is centered around the origin. To speed up the optimization and to avoid poor local minima, a large momentum term is added to the gradient. In other words, to find the changes in the coordinates of the map points at every iteration of the gradient search, the present gradient is added to an exponentially decaying sum of preceding gradients. Mathematically, the update of the gradient with respect to momentum term is given by

$$\mathscr{Y}^{(t)} = \mathscr{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathscr{Y}} + \alpha(t)(\mathscr{Y}^{(t-1)} - \mathscr{Y}^{(t-2)}), \tag{1.10}$$

where $\mathscr{Y}(t)$ denotes the solution at iteration t, η represents the learning rate, and $\alpha(t)$ represents the momentum at iteration t.

At the starting point of the optimization, after every iteration the Gaussian noise is added to the map points. In order to reduce the variance of this noise gradually, a type of simulated annealing is performed that helps the optimization to escape from poor local minima in the cost function. In case the variance of the noise changes gradually at the critical point at which the global structure of the map is formed, SNE is likely to find maps with a good global organization. But this requires very sensible choices of initial Gaussian noise and the rate at which it decays. Therefore, it is common to run the optimization many times on a data set to find the appropriate values for the parameters.

Inspite of SNE constructing reasonably better visualizations, it is hindered by a cost function that is difficult to optimize and this problem is referred to as the "crowding problem". A new technique called "t-Distributed Stochastic Neighbor Embedding" or "t-SNE" that aims to eradicate these problems.
The cost function used by t-SNE varies from SNE in two ways. Firstly, it utilizes the symmetrized version of the SNE cost function with uncomplicated gradients. Secondly, it utilizes the Student - t distribution in place of Gaussian to calculate the similarity between two points in the low-dimensional space. In order to reduce the crowding problem and the optimization problems of SNE, t-SNE makes use of a heavy-tailed distribution in the low-dimensional space.
Further, we discuss symmetric SNE, crowding problem, and the use of heavy-tailed distributions.

1.2.1 Symmetric SNE

Rather than reducing the sum of Kullback-Leibler divergences between the conditional probabilities $P_{j|i}$ and $q_{j|i}$, it is also possible to reduce the individual Kullback-Leibler divergence between a joint probability distribution, P, in the high-dimensional space and a joint probability distribution, Q, in the low-dimensional space [8]: $C = KL(P||Q)$

$$C = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{1.11}$$

where we again set p_{ii} and q_{ii} to zero. We mention this type of SNE as symmetric SNE, due to the property that $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}\ \forall i, j$ [8].
The pairwise similarities of symmetric SNE in the low-dimensional map q_{ij} are given by [8]

$$q_{ij} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq l} exp(-||y_k - y_l||^2)} \tag{1.12}$$

The apparent way to define the pairwise similarities in the high-dimensional space p_{ij} is [8]

$$p_{ij} = \frac{exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k \neq l} exp(-||x_k - x_l||^2/2\sigma^2)}, \tag{1.13}$$

still this causes issues when a high-dimensional data point x_i is an outlier (i.e we get larger pairwise distances $||x_i - x_j||^2$ for x_i).
For the corresponding outlier, the values of p_{ij} are very small for all j, hence the location of y_i in the low-dimension has a very small effect on the cost function [8].
To resolve this mapping problem we define the joint probabilities p_{ij} in the high-dimensional space to be the symmetrized conditional probabilities, that is given by [8]

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{1.14}$$

This makes sure that $\sum_j p_{ij} > \frac{1}{2n}$ for all data points x_i, due to this every data point x_i makes a important contribution to the cost function.
The gradient of symmetric SNE are simpler, faster to compute and is reasonably alike to asymmetric SNE, that is given by [8]

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j) \tag{1.15}$$

1.2.2 Crowding problem of SNE

It is the problem raised when the area of the two dimensional map that is available to take in moderately distant datapoints will not be big enough compared with the area available to take in nearby datapoints [8]. This problem arises when the datapoints that are distributed in the area of the high-dimensional manifold around i, and we try to model the pairwise distances from i to the datapoints in a two-dimensional map [8].
Suppose in case it is feasible to have 11 datapoints that are mutually equidistant in a ten-dimensional manifold but it is not feasible to model this truly in a two-dimensional map [8]. Hence, if we can model correctly small distances data points in a map, then most of the moderately distance datapoints will be very far away in the two-dimensional map. In SNE, this will develop a small attractive force from datapoint i to these very far distant map points [8]. The large number of such forces together breaks down the points in the center of the map and avoids gaps between the natural clusters. An attempt has been made by Cook et al.(2007) to resolve the crowding problem, by adding a slight repulsion to all springs [8]. The slight repulsion is built by using a uniform background model with a small mixing proportion ρ, that helps q_{ij} to never go below $\frac{2\rho}{(n)(n-1)}$ [8].

1.2.3 Use of Heavy-tailed distribution by t-SNE

The symmetric SNE matches the joint probabilities between the pair of datapoints in high-dimensional and low-dimensional datapoints rather than its distances. There is a natural way to resolve this crowding problem. We start converting distances into probabilities in the high-dimension space using a Gaussian distribution [8]. Later we use the probability distribution with heavy-tails in comparison to Gaussian to convert distances into probabilities in low-dimensional space [8]. This makes it possible to allow moderate distant datapoints in the high-dimensional space to be truly modeled by a much larger distance in the map. As a result of this we can avoid the undesirable attractive forces between the datapoints in the map that represents moderately dissimilar datapoints [8].

In t-SNE, in case of low-dimensional map we use student t-distribution with one degree of freedom as the heavy-tailed distribution [8]. Making use of this distribution, the joint

probabilities q_{ij} are defined as [8]

$$q_{ij} = \frac{(1+||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1+||y_k - y_l||^2)^{-1}} \tag{1.16}$$

The student t-distribution with a single degree of freedom is used because it holds a good property that $(1+||y_i - y_j||^2)^{-1}$ reaches an inverse square law for large pairwise distances $||y_i - y_j||$ in the low-dimensional map [8]. The inverse square law can be interpreted as the value of the inverse of the square of the map points y_i and y_j does not affect the value of the joint probabilities [8].
This creates the map's representation of joint probabilities q_{ij} relatively unchanged in the scale of low-dimensional space for datapoints that are far apart [8].
The theoretical justification for using Student t-distribution is that it nearly resembles to the Gaussian distribution, as it is an infinite mixtures of Gaussians [8]. The density of a point under student t-distribution can be evaluated much faster in comparison to Gaussian as it does not involve an exponential [8]. The gradient of the Kullback-Leibler divergence linking P and the Student-t based joint probability distribution Q and is given by [8]

$$\frac{\delta C}{\delta y_j} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)(1+(||y_i - y_j||)^2)^{-1} \tag{1.17}$$

There are two main merits of the t-SNE gradient in comparison to the gradients of SNE [8]. First, the t-SNE gradient firmly repels dissimilar data points which are modeled in the low-dimensional representation by a small pairwise distance [8]. Second, although t-SNE introduces repulsions vigorously between dissimilar data points which are modeled by small pairwise distances and the repulsions do not go to infinity [8].

1.2.4 Parameters used in t-SNE

The important hyperparameter used in t-SNE is perplexity [8]. The perplexity is linked to the nearest neighbors used in manifold learning algorithms [10]. The value of perplexity would be more for larger datasets [10]. Different perplexity values will give significantly different results [10]. Generally, the values of perplexity used is between 5 and 50. Under different settings of the perplexity, the performance of the t-SNE is quite robust [8]. The optimization parameters are number of iterations, learning rate [8]. Suppose the learning rate is high, the data will look like a 'ball' with any point approximately equidistant from its nearest neighbors [10]. If learning rate is low, then most points may look compressed in a dense cloud with fewer outliers [10].

2 Evaluation of the visualization methods

The evaluation is based on supervised learning algorithm and k-means clustering.

2.1 Supervised learning algorithm

This section is based on the book from A. Patel [9]. Supervised methods consist of two major problems: classification and regression. Classification problems are also called as discrete prediction problems as each class is a discrete group. They are also referred to as qualitative or categorical problems. Regression must predict a continuous variable and also refer to as quantitative problems.
Supervised machine learning algorithms from simple to complex, are aimed at minimizing cost function or error rate which is associated with the labels we have for the dataset. The choice of a supervised learning algorithm is important for minimizing generalization error. The lowest possible generalization error can be achieved if the complexity of the algorithmic model matches the complexity of the true function underlying the data.
We have no idea about the true function, so we make use of machine learning to create a model and solve the function to find the correct solution. In case the algorithm models are less complex than the true function, then we have underfit the data. So in such cases, we would improve the generalization error by choosing an algorithm that can model a more complex function. But if the algorithm creates an overly complex model, it will overfit the training data and give a poor performance on unseen cases, which in turn increases the generalization error. Hence, it is not necessary to choose more complex algorithms over simpler ones, sometimes simpler algorithms are a good choice.
Some of the methods for supervised learning would be linear methods, neighborhood-based methods, tree-based methods, support vector machines. For our experiments, we use a tree-based method that is Random forests. Random forests are used to improve overfitting by sampling the instances and predictors. By using random forests, we consider multiple random samples of instances from the training data. Also in case of predictors, we do not split all of them instead we choose a random sample of the predictors. The square root of the total number of predictors is used in choosing each split of the number of predictors. By this way of sampling of the predictors, the random forests algorithm makes trees which are less correlated and reduces the overfitting and improves the generalization error.
The hyperparameters used for random forests are a number of estimators which basically is used to build the trees and average the results across the trees. Now in each tree, the model considers the square root of the total number of features. We set max depth parameter to none, for the tree to grow as deep as possible with the given subset of features.
But using random forest, one of the tree-based methods, we make classification reports

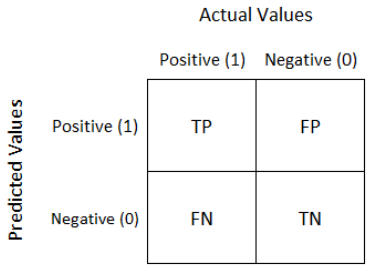

Figure 2.1: Confusion matrix

and confusion matrix. Here we also use Principal Component Anlaysis (PCA) as a tool for dimensionality reduction. The evaluation is done after we train the model with non-linear models autoencoders, PCA, and t-SNE. With non-linear methods, labels are not used for training data instead they are used for evaluation purpose with random forest classifiers. The dimension of the dataset is reduced by the non-linear methods and the reduced dimension (encoded data) that is code size from the autoencoders is given to the classifier. The reduced dimension from PCA and t-SNE is also further used and visualized in two and three dimensions and evaluated.
We use confusion matrix, F1-measure, precision, recall for finding the accuracy with micro, macro, and the weighted average for evaluating the results from the non-linear visualization methods. We consider the evaluation metrics to help us more intuitively understand the results. Confusion matrix used for evaluating the results is a table that summarizes the number of true positives, true negatives, false positives, and false negatives as shown in Figure 2.1.

For balanced classes, the number of true positives roughly similar to the number of true negatives, the confusion matrix is good and straightforward metric.
To evaluate in a better manner, we use metric precision, recall, F1-measure.
Precision = TP/ TP + FP
Recall = TP/ TP + FN
F1 = 2TP/ 2TP + FP + FN

where TP, FP, TN, FN represent the counts for true positive, false positive, true negative and false negative, respectively.

2.2 K-means clustering

For clustering, we identify distinct groups in such a way that instances within a group are similar to each other and dissimilar in other groups [13]. K-means clustering is one such algorithm which helps in clustering data and to separate samples in n groups of equal variance, reducing a criterion called the inertia or within-cluster sum-of-squares [13]. In this algorithm, we need to specify the number of clusters [13]. It scales the large samples and can be used for large range of applications in different fields [13].
The k-means algorithm splits a set of N Samples X to K disjoint clusters C, which is described by the mean μ_j of the samples in the clusters [13]. The mean of the clusters is often called the centroids, in general, they are not the points from X, even though they live in same space [13].
The k-means chooses centroids to reduce inertia or minimize the within-cluster sum-squares criteria [13]

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_i - \mu_j||^2) \tag{2.1}$$

The algorithm has three basic steps, the first step involves choosing the initial centroids and choosing samples from the dataset which are called the training data [13]. After this basic initialization step, K-means consists of looping between the remaining two steps [13]. The first step involves the assignment of each sample to its nearest centroid [13]. The second step deals with creating new centroids by taking the mean value of all the samples assigned to every earlier centroid [13]. The difference between the previous and the newly created centroids are calculated and the algorithm repeats the last two steps until this value is less than a threshold [13]. Alternatively, it repeats until the centroids do not change anymore [13]. The algorithm can be understood better by Voronoi diagrams with clusters and centroid marked as shown in Figure 2.2 [13]. Initially using current centroids, the Voronoi diagram of the points is computed. A separate cluster is formed in every segment of the Voronoi diagram [13]. Next, for the mean of every segment, the centroids are updated [13]. The algorithm repeats this until the stopping criterion is satisfied [13].

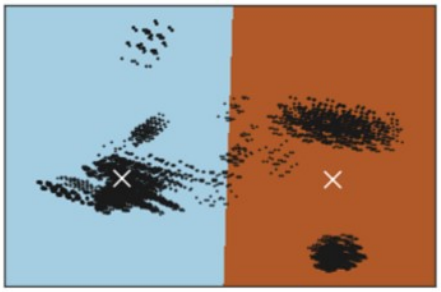

Figure 2.2: Voronoi diagram with clusters (centroids are marked with white cross)

K-means will converge if given sufficient time, perhaps this may be to a local minimum. Again this highly depends on the initialization of the centroids. One way to solve this issue is the k-means++ initialization scheme, which we have implemented using scikit-learn [10]. Generally, this initializes the centroids to be away from each other, which leads to better results in comparison to random initialization [10].
The algorithm also supports a parameter called sample weight. This helps to assign more weight to a few samples when calculating cluster centers as well as values of inertia [10].
The performance evaluation for clustering is not as insignificant as by counting the number of errors or the precision and recall of a supervised classification algorithm [10]. Specifically, any evaluation metric should not consider the absolute values of the cluster labels into account [10]. Clustering defines separations of the data related to some ground truth set of classes or fulfilling some expectation that members of the same class are more similar than in different classes as per some similarity metric as discussed below [10].

2.2.1 Adjusted Rand index

Considering the ground truth class assignments (true labels) and clustering algorithm assignments of the same samples (predicted labels), the function Adjusted Rand index measures the similarity between the two assignments and ignores the permutations with chance normalization [10].
For example, the labels are 0, 1, and 2, with the permutations of labels 0 and 1 and renaming of labels from 2 to 3 will not change the Adjusted Rand index score [10]. We consider C as a ground truth class assignment and K as clustering, further we define a and b as:
a as the number of pair of elements which are in same set in C and as well in same set in K and b as the number of pair of elements which are in different sets in C and as well in different sets in K.
The raw (unadjusted) Rand index is then given by [10]:

$$\mathrm{RI} = \frac{a+b}{C_2^{n_{samples}}} \tag{2.2}$$

Where $C_2^{n_{samples}}$ is referred as total number of possible pairs in the dataset (without ordering).
Although the RI score does not assure that random assignments will receive a value close to zero (particularly if the number of clusters are in the same order as that of magnitude of the samples) [10].
To tackle this effect we can take the expected RI ($E[\mathrm{RI}]$) of random labelings by defining

the Adjusted Rand index as follows [10]:

$$\text{ARI} = \frac{\text{RI} - E[\text{RI}]}{\max(\text{RI}) - E[\text{RI}]} \tag{2.3}$$

The advantages of ARI score is: the score will be close to 0.0 for random label assignments for any values of the number of clusters and number of samples [10]. The bounded range is [-1,1]: usually, the negative values are bad, positive ARI have similar clusterings, the perfect match score for ARI will be 1.0 [10].

2.2.2 Mutual Information based scores

Considering again the knowledge of the ground truth class assignments and with cluster algorithm assignments of the same samples, the Mutual Information is a function that measures the consensus of the two assignments, ignoring permutations [10]. There are two different normalized versions of this measure, Normalized Mutual Information (NMI) and Adjusted Mutual Information (AMI) [10].
Let us assume two label assignments (suppose of same N objects), U and V. We can define their entropy which is the amount of uncertainty for a partition set, defined by [10] :

$$H(U) = -\sum_{i=1}^{|U|} P(i)\log(P(i)) \tag{2.4}$$

where $P(i) = |U_i|/N$ is the probability of picking an object at random from U which falls into class U_i.
Likewise for V:

$$H(V) = -\sum_{j=1}^{|V|} P'(j)\log(P'(j)) \tag{2.5}$$

With $P'(j) = |V_j|/N$. The mutual information (MI) between U and V is calculated by [10]:

$$\text{MI}(U,V) = \sum_{i=1}^{|U|}\sum_{j=1}^{|V|} P(i,j)\log\left(\frac{P(i,j)}{P(i)P'(j)}\right) \tag{2.6}$$

where $P(i,j) = |U_i \cap V_j|/N$ is the probability of picking an object at random which falls into both classes U_i and V_j.
It can also be expressed in set cardinality formulation: [10]

$$\text{MI}(U,V) = \sum_{i=1}^{|U|}\sum_{j=1}^{|V|} \frac{|U_i \cap V_j|}{N}\log\left(\frac{N|U_i \cap V_j|}{|U_i||V_j|}\right) \tag{2.7}$$

The normalized mutual information is defined as [10] :

$$\text{NMI}(U,V) = \frac{\text{MI}(U,V)}{\text{mean}(H(U),H(V))} \tag{2.8}$$

The value of mutual information obtained and also the normalized variant is not adjusted for chance and tends to increase as the clusters increase, in spite of the actual amount of "mutual information" between the label assignments [10].
The below equation can be used to compute the expected value for the mutual information [14].
In this equation, $a_i = |U_i|$ and $b_j = |V_j|$.

$$E[\text{MI}(U,V)] = \sum_{i=1}^{|U|}\sum_{j=1}^{|V|}\sum_{n_{ij}=(a_i+b_j-N)}^{\min(a_i,b_j)} \frac{n_{ij}}{N}\log\left(\frac{Nn_{ij}}{a_i b_j}\right) \frac{a_i!b_j!(N-a_i)!(N-b_j)!}{N!n_{ij}!(a_i-n_{ij})!(b_j-n_{ij})!(N-a_i-b_j+n_{ij})!} \tag{2.9}$$

Using the expected value, the adjusted mutual information can then be computed using a similar form to that of the adjusted Rand Index [10]:

$$\text{AMI} = \frac{\text{MI} - E[\text{MI}]}{\text{mean}(H(U),H(V)) - E[\text{MI}]} \tag{2.10}$$

The Adjusted Mutual Information (AMI) score will be close to 0.0 for random label assignments for any values of the number of clusters and number of samples (generally not in case of raw Rand Index) [10].
The values of AMI score that are close to zero indicates two label assignments which are largely independent [10]. The values that are close to one indicates significant consensus. Additionally, an AMI with a precise value of 1 indicates that the two label assignments are equal (with or without permutation) [10].

2.2.3 Homogeneity, completeness and V-measure

Considering the knowledge of the ground truth class assignments of the samples, we can define some instinctive metric using conditional entropy analysis [10].
Specifically, Rosenberg and Hirschberg (2007) define the following two useful objectives for any cluster assignment, Homogeneity as every cluster contains only members of individual class and Completeness as every member of the given class is assigned to the same cluster [10].
These concepts are termed as scores: homogeneity score and completeness score [10]. Both scores are bounded below by 0.0 and above by 1.0 (higher is better).

Their harmonic mean is called V-measure that is calculated by v measure score [15]: This function's formula is as follows:

$$v = \frac{(1+\beta) \times \text{homogeneity} \times \text{completeness}}{(\beta \times \text{homogeneity} + \text{completeness})} \tag{2.11}$$

beta defaults to a value of 1.0.
Homogeneity and completeness scores are precisely given by [10]:

$$h = 1 - \frac{H(C|K)}{H(C)} \tag{2.12}$$

$$c = 1 - \frac{H(K|C)}{H(K)} \tag{2.13}$$

where $H(C|K)$ is the conditional entropy of the classes given the cluster assignments which is defined as:

$$H(C|K) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{n_{c,k}}{n} \cdot \log\left(\frac{n_{c,k}}{n_k}\right) \tag{2.14}$$

and $H(C)$ is the entropy of the classes and is defined as:

$$H(C) = -\sum_{c=1}^{|C|}\frac{n_c}{n} \cdot \log\left(\frac{n_c}{n}\right) \tag{2.15}$$

with n the total number of samples, n_c and n_k the number of samples individually belonging to class c and cluster k, and finally $n_{c,k}$ the number of samples from class c assigned to cluster k. The conditional entropy of clusters given class $H(K|C)$ and the entropy of clusters $H(K)$ are defined in a symmetric manner.
Rosenberg and Hirschberg defined V-measure as the harmonic mean of homogeneity and completeness [10]:

$$v = 2 \cdot \frac{h \cdot c}{h + c} \tag{2.16}$$

The advantages of V-measure are bounded scores: 0.0 is a very bad score possible, whereas 1.0 is a perfect score [10]. The clustering with bad V-measure is analzed in phase of homogeneity and completeness to get an overview of the different mistakes done by the assignment [10].

2.2.4 Silhouette Coefficient

Suppose the ground truth labels are not known, evaluation is performed from the model itself [16]. For example, the Silhouette Coefficient is such an evaluation, where a higher Silhouette Coefficient score relates to a model with well-defined clusters [16]. The Silhouette Coefficient is defined for each sample and is consists of two scores: Firstly, the mean distance between a sample and the remaining other points in the same class. Secondly, the mean distance between a sample and remaining other points in the next nearest cluster [10].

The Silhouette Coefficient s for a single sample is given as [10]:

$$s = \frac{b - a}{max(a, b)} \tag{2.17}$$

The advantages of Silhouette Coefficient score are, the score is bounded between -1 and +1, -1 is for incorrect clustering and +1 for highly dense clustering [10]. Overlapping clusters are indicated with score zero. The score is higher when clusters are dense and well separated, which relates to a standard notion of a cluster [10].

3 Experiments

The datasets used for nonlinear visualization for the experiments are breast cancer dataset and mushroom dataset.

3.1 Breast cancer dataset

The breast cancer dataset contains features that are computed from a digitized image of a fine needle aspiration (FNA) of a breast mass. The characteristics of cell nuclei present in breast mass are described in the dataset. The characteristic of the dataset is multivariate or bivariate as their data values are ordered pairs of data [12].
The characteristics of attributes are real or metric and in total there are 32 number of attributes with 569 number of instances. The attributes contain ID number, diagnosis (M = malignant, B = Benign) and the remaining ten real-valued features radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, fractal dimensions are used to compute each cell nucleus [12].
The dataset contains real or metric values useful for numeric calculations but the diagnosis variable contains categorical values. The id column of the dataset is dropped and we convert the diagnosis column to numeric format using factorize function on the dataframe. There is some unnamed variable found in the dataset that contains null values which are filled with a value zero using the function fillna from pandas.
The diagnosis column now contains numeric values and they describe the malignant and benign condition of the cells. These describe the class labels but we drop this column and store it in variable Y and we use label encoder, fit and transform with new encoded data ranging between 0 and (number of classes) - 1.
Then we use the standard scalar function from sklearn library on variable X which contains all feature values for training and for standardizing features by removing the mean and scaling unit variance. Finally the data variable X with 31 attributes (high-dimensional input data) is reduced to 2 dimensional data for visualization using autoencoders and t-SNE.

3.2 Mushroom dataset

Mushroom records are drawn from the Audubon society field guide to North American mushrooms(1981), G.H Lincoff, NewYork: Alfred A. Knopf. The records describe mushroom's physical characteristics and classification of whether it is poisonous or edible [11]. The characteristics of the dataset are multivariate or bivariate as their data values are ordered pairs of data. The attribute characteristics are categorical and there is a

total 23 number of attributes with 8124 number of instances. The dataset includes hypothetical samples corresponding to 23 species of gilled mushrooms [11].
Each species can be identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. We basically count the values of edible and poisonous mushrooms from the dataframe attribute class. All the categorical attributes are stored in data variable X and we drop the column class from X and store it in variable Y which is later used for evaluation purposes. Later we introduce get dummies function to convert categorical variable of 'X' into model-understandable dummy/indicator variables (numeric data with 117 dimensions).
Then we use label encoder class from sklearn library, fit and transform the class labels stored in variable Y and we convert this target labels with new encoded data ranging between 0 and (number of classes) - 1. Then we use Standard Scalar function on variable X before the training begins for standardizing features by removing the mean and scaling to unit variance. Centering and scaling occurs independently on each feature by computing various statistics on the samples in the training set. The standard deviation and mean are then stored to use it later on data using transform.

3.3 Evaluating basic autoencoders based on supervised learning algorithms

We start our experiment by considering breast cancer dataset and obtain visualizations using basic autoencoders in two dimension. We also experiment with loss function like binary crossentropy and optimizers like adam, and adadelta. The adadelta optimizer uses stochastic gradient descent with adaptive learning rate [10].
We start building basic autoencoders with input layer assigned with 31 dimensions of input X which is high dimension. Next, we use two hidden layers with layer size equal to 10 to extract the abstract features. The hidden representations in low dimension are viewed in two dimension used to visualize the high dimensional data. The output layer from the network maps the hidden layer representation back to a reconstruction input. The basic autoencoder is modeled with input and output and then compiled further with adadelta optimizer and binary cross entropy loss function. The training and testing split of the data is done for the input data and training the model begins by assigning the epochs, batch size and verbose levels. Finally, the low dimensional data that is two dimensional data with input layer is modeled and we start predicting and classify the input data. The prediction is done by using the Confusion matrix as shown in Table 3.1. As observed from confusion matrix, the number of true negative patients are 96 and 56 true positive. We find the classification report for the class labels of the dataset using the random forest with the metric Precision, Recall, and F1-score for accuracy as shown in Table 3.2.
The Visualization of basic autoencoder with adadelta optimizer and binary crossentropy loss function as shown in Figure 3.1.

		Actual Values	
		Positive	Negative
Predicted Values	Positive	56	10
	Negative	09	96

Table 3.1: Confusion matrix for the prediction of basic autoencoder

	Precision	Recall	F1-score
0	0.86	0.85	0.85
1	0.91	0.91	0.91
micro avg	0.89	0.89	0.79
macro avg	0.88	0.88	0.78
weighted avg	0.89	0.89	0.79

Table 3.2: Evaluation results of basic autoencoder with adadelta optimizer

Next, the basic autoencoder with adam optimizer and Kullback Leibler divergence loss function is built in similar manner as described above. The confusion matrix for the prediction of basic autoencoder is as shown in Table 3.3. Further, the evaluation of classification reports as shown in Table 3.4 and visualization of breast cancer dataset using basic autoencoders can be seen in Figure 3.2. The use of adam optimizer gives slightly better results than adadelta optmizer as observed in visualization and classification reports due to its optimization algorithm for stochastic gradient descent for training deep learning models.

		Actual Values	
		Positive	Negative
Predicted Values	Positive	57	9
	Negative	9	95

Table 3.3: Confusion matrix for the prediction of basic autoencoder

	Precision	Recall	F1-score
0	0.86	0.86	0.86
1	0.91	0.91	0.91
micro avg	0.89	0.89	0.79
macro avg	0.89	0.89	0.79
weighted avg	0.89	0.89	0.79

Table 3.4: Evaluation results of basic autoencoder with adam optimizer

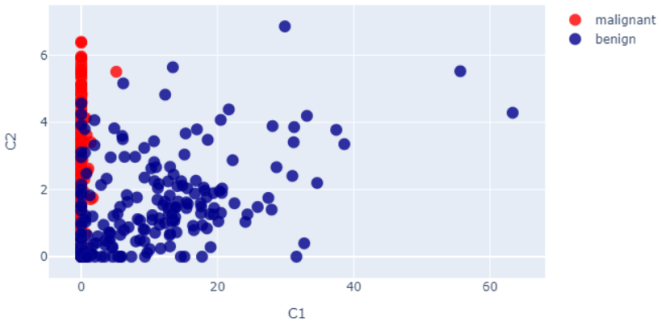

Figure 3.1: Visualization of breast cancer dataset by basic autoencoder (adadelta optimizer)

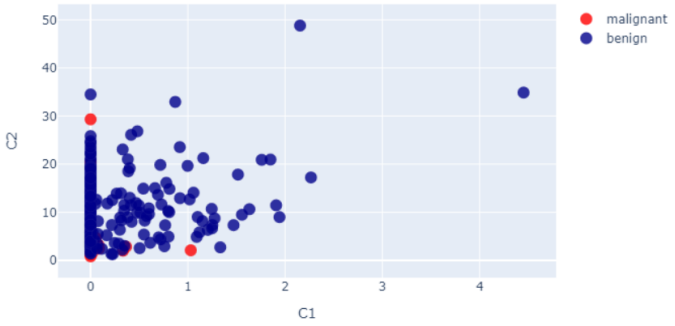

Figure 3.2: Visualization of breast cancer dataset by basic autoencoder (adam optimizer)

3.4 Evaluating Deep autoencoders based on supervised learning algorithms

We build deep autoencoders in similar manner with input layer assigned with 31 dimensions of input X but with additional hidden layers, code layer, encoder, and decoder. The additional two hidden layers with layer size equal to 20 and 14 are used in deep autoencoders helps in extracting more abstract features in comparison to basic autoencoders. The extra hidden layers also help in better reconstruction of the input with minimum loss. The code layer in between the hidden layers is the main hyperparameter that we fixed as two for visualization of the breast cancer dataset. The deep autoencoder is modeled with input and output and then compiled further with adam optimizer and mean square error loss function. The training and test data splitting is same as basic autoencoders. Finally, the two-dimensional data with the input layer is modeled and we start predicting and classify the input data using a random forest classifier. The prediction is done by using the Confusion matrix as shown in Table 3.5. As observed from confusion matrix, the number of true negative patients are 91 and 58 true positive. We find the classification report for the class labels of the dataset using the random forest with the metric Precision, Recall, and F1-score for accuracy as shown in Table 3.6 and visualization in Figure 3.3. The results of deep autoencoders in comparison to basic autoencoders are better with additional hidden layers.

		Actual Values	
		Positive	Negative
Predicted Values	Positive	58	8
	Negative	14	91

Table 3.5: Confusion matrix for the prediction of Deep autoencoder

	Precision	Recall	F1-score
0	0.91	0.91	0.89
1	0.92	0.92	0.92
micro avg	0.91	0.91	0.92
macro avg	0.92	0.91	0.92
weighted avg	0.92	0.91	0.92

Table 3.6: Evaluation results of Deep autoencoder

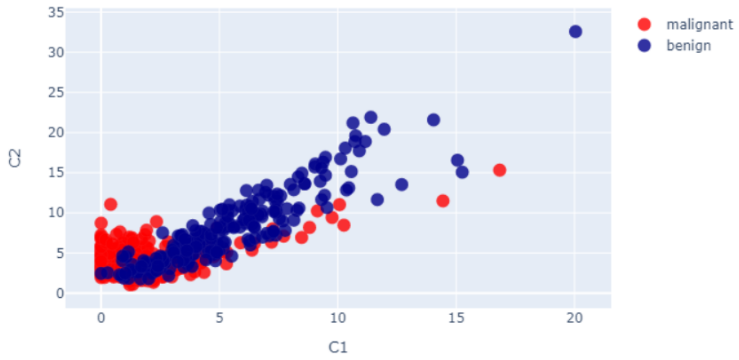

Figure 3.3: Visualization of breast cancer dataset by deep autoencoder

3.5 Evaluating PCA, and t-SNE based on supervised learning algorithms

We build PCA by considering our input data which is high-dimensional data and further through fit and transform function it is converted into numeric format. Then for visualization we use the parameter of PCA , the number of components (PC1 and PC2) that we fix to two. The two components are the two dimensions and with the output labels the complete test with random forest classifier is used for evaluation of PCA. The confusion matrix for the prediction of PCA is as show in Table 3.7 and evaluation of PCA through the metric Precision, Recall and F1-score is as shown in Table 3.8. The visualization of the breast cancer dataset with malignant and benign classes is shown in Figure 3.4. Though PCA is just used to see the variance in the data captured by the PCA components, we get 0.6324 variance for first 10 components and 0.9955 variance for 20 components.

		Actual Values	
		Positive	Negative
Predicted Values	Positive	60	6
	Negative	8	97

Table 3.7: Confusion matrix for the prediction of PCA

	Precision	Recall	F1-score
0	0.88	0.91	0.90
1	0.94	0.92	0.93
micro avg	0.92	0.92	0.92
macro avg	0.91	0.92	0.91
weighted avg	0.92	0.92	0.92

Table 3.8: Evaluation results of PCA

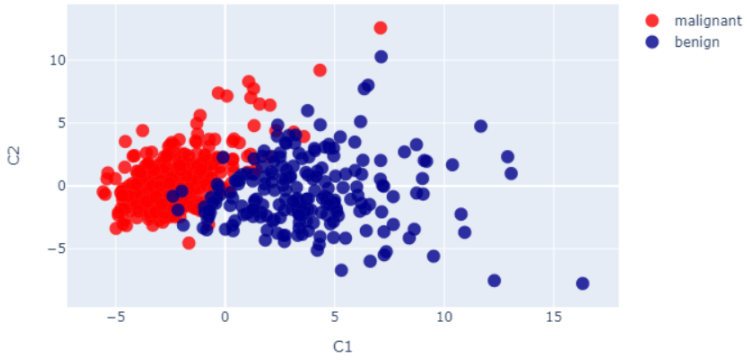

Figure 3.4: Visualization of breast cancer dataset by PCA

Finally, we evaluate t-SNE to visualize the breast cancer dataset. We use the parameters of t-SNE that is number of component which is fixed to two for low-dimensional visualization. The next important parameter for visualization is perplexity which is the measure of effective number of neighbors of the samples. We set this parameter initially with 30, 40 and finally 50. The two dimensional visualization with the various values of perplexities can be seen from the Figures 3.5, 3.6, and 3.7. The visualizations changes with the change in the perplexity as the effective number of neighbors also gets affected. Then the t-SNE algorithm computes the conditional probabilities for the neighbors of the data and at the end it provides a mean standard deviation (variance) value with KL divergence with 300 iteration used for the experiment. In the similar manner we can see the confusion matrix in the Table 3.9 for the prediction of t-SNE and the evaluation of t-SNE with the random forest classifier in the Table 3.10.

		Actual Values	
		Positive	Negative
Predicted Values	Positive	61	5
	Negative	2	103

Table 3.9: Confusion matrix for the prediction of t-SNE

	Precision	Recall	F1-score
0	0.97	0.92	0.95
1	0.95	0.87	0.97
micro avg	0.96	0.96	0.96
macro avg	0.96	0.95	0.96
weighted avg	0.96	0.96	0.96

Table 3.10: Evaluation results of t-SNE

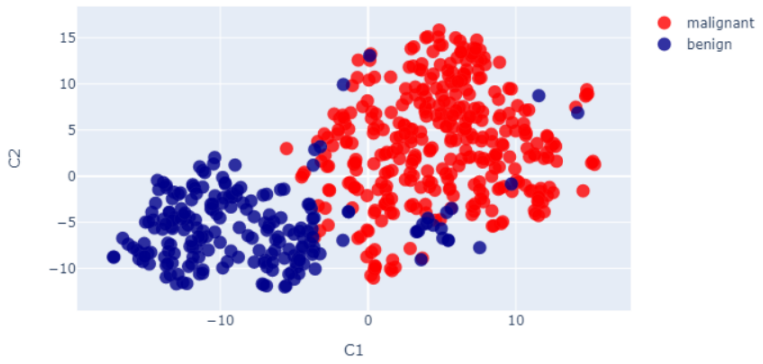

Figure 3.5: Visualization of breast cancer dataset by t-SNE with Perplexity=30

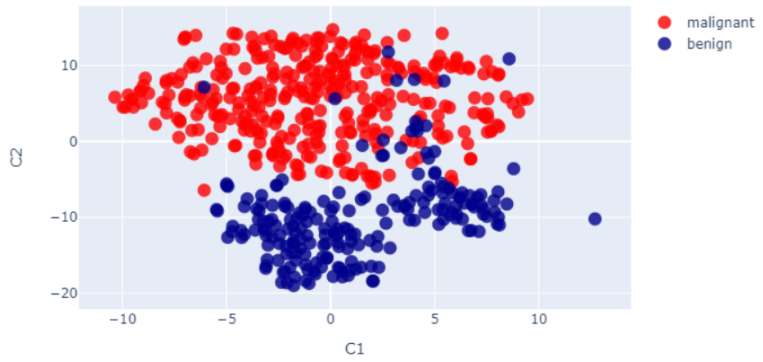

Figure 3.6: Visualization of breast cancer dataset by t-SNE with Perplexity=40

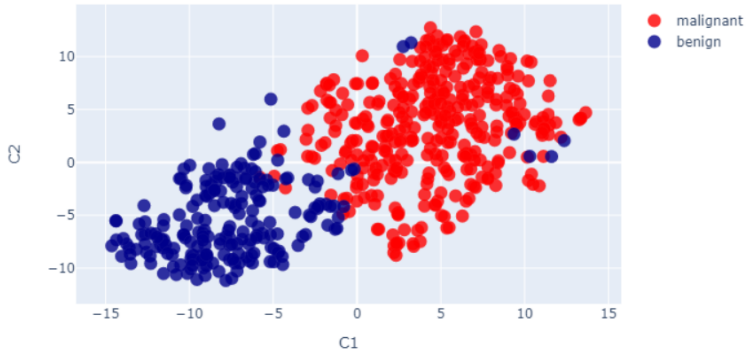

Figure 3.7: Visualization of breast cancer dataset by t-SNE with Perplexity=50

3.6 Choosing and implementing various cost functions for autoencoders

Deep learning neural networks use stochastic gradient descent optimization algorithm for training [19]. With the optimization algorithm, the error for the present state of the model is estimated repeatedly. For this we need to choose an error function, called as loss function, which is used to estimate the loss of the model such that the weights can be updated in order to reduce the loss on the next evaluation [19].
Neural network models learn to map from input to output and the choice of loss function must match the specific predictive modeling problem, such as classification or regression [19]. Further, it must be an appropriate loss function for the output layer configuration.
We are using mushrooms dataset for our experiment based on different cost function like Mean Squared Error Loss, Binary Cross-Entropy, Hinge Loss on various autoencoders like basic autoencoder with and without hidden layers and deep autoencoders.
The default loss to use for regression problems is the mean squared error or MSE. Mathematically, MSE is preferred loss function under the framework of maximum likelihood if the target variable distribution is Gaussian [19]. Mean squared error can be calculated as the average of the squared differences between the actual and the predicted values [19]. The result is positive nevertheless of the sign of the predicted and the actual values, and a perfect value is 0.0. The squaring indicates that larger mistakes give more error than smaller mistakes, that is model is punished for making bigger mistakes.
We have created a line plot in showing the mean squared error loss over the training epochs with training (blue) and testing (orange) sets in Figure 3.8 and 3.9. We tested with basic and deep autoencoders and we got slightly better classification accuracy with deep autoencoders. We can observe that the model converged reasonably quickly and both train and test performance remained the same. The performance and convergence of the model suggest us that mean squared error is a better match for deep autoencoders.

Another way to cross-entropy for binary classification problems is the hinge loss function, which is developed mainly for Support Vector Machine(SVM) models [19]. It is mainly used with binary classification where we set the target values to -1,1 [19]. The hinge loss function helps the example dataset to have the correct sign, giving more error when there is a larger difference between the actual and predicted class values. Finally, the output layer of the network is configured with a hyperbolic tangent activation function which is capable of giving a single value output in the range [-1, 1].
The performance reports with the hinge loss are mixed, resulting in better performance sometimes than cross-entropy for binary classification problems. The two-line plots are created as shown in Figure 3.10 and 3.11 , the top being the hinge loss over epochs for the train (blue) and test (orange) dataset, and the bottom plot shows classification accuracy over epochs. The plot of hinge loss for the basic and deep autoencoders has converged with a reasonable loss. But the plot of classification accuracy over training epochs does not show sign of convergence.
Next we use default loss function Cross-entropy to solve binary classification problems [19]. It is mainly used where the target values are in the set $\{0, 1\}$. Cross-entropy calculates and summarizes the average difference between the actual and the predicted probability distributions for predicting class 1 [19]. Further, the score is minimized with a perfect cross-entropy value 0.
The function requires the output layer to be configured with sigmoid activation to predict the probability for class 1. The two-line plots are created, the top with the cross-entropy loss over epochs for the training (blue) and testing (orange) dataset, and the bottom plot with classification accuracy over epochs.
The plot in Figure 3.12 indicates that the training process has converged and the plot for loss is smooth for deep autoencoders. The line plot for accuracy shows better accuracy in comparison to other loss functions.

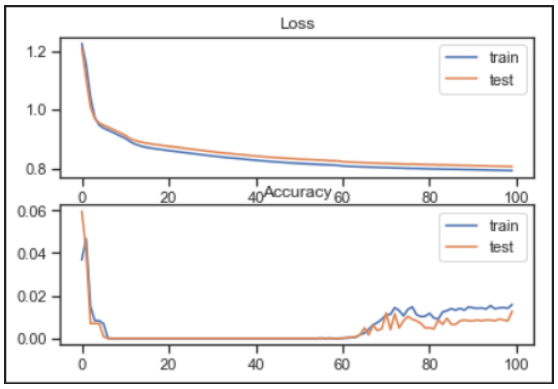

Figure 3.8: Basic autoencoders Line plot of Mean squared error Loss and classification accuracy over training Epochs

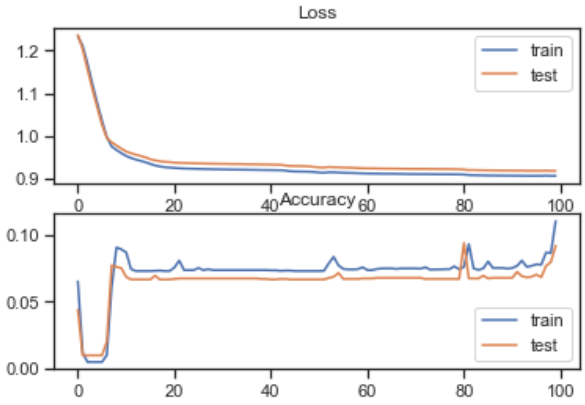

Figure 3.9: Deep autoencoders line plot of Mean squared error loss and classification accuracy over training epochs

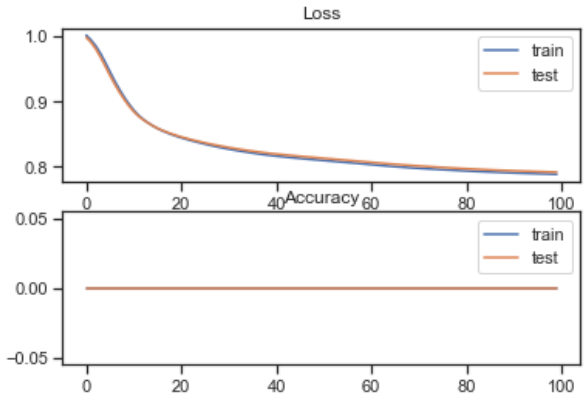

Figure 3.10: Basic autoencoders, Line plot of hinge loss and classification accuracy over training epochs

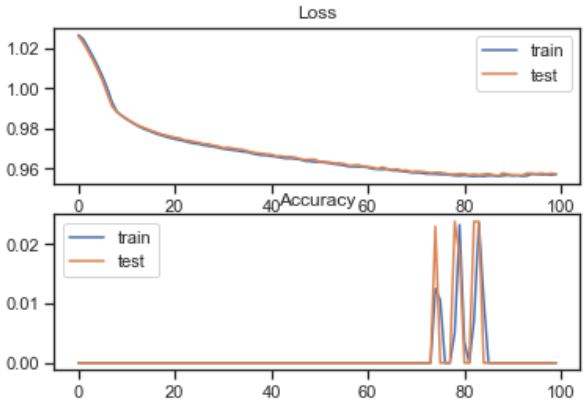

Figure 3.11: Deep autoencoders line plot of Hinge Loss and classification accuracy over training epochs

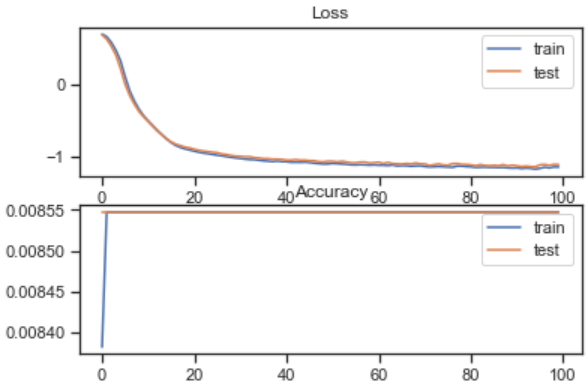

Figure 3.12: Deep autoencoders Line plot of Binary cross entropy loss and classification accuracy over training epochs

3.7 Evaluating autoencoders, t-SNE based on K-means clustering

The quality of the clustering is calculated using the metrics Homogeneity score (homo), Completeness score (compl), V measure (v-meas), Adjusted rand index (ARI), Adjusted mutual information (AMI) and Silhouette coefficient (silhouette). The scores of the metrics obtained by PCA, autoencoders, t-SNE are all within the range [0,1] and the large score indicates to the point that well belongs to the cluster. The mushroom dataset is used and classified using the k-means clustering algorithm and visualized using the dimensionality reduction technique PCA, autoencoders, t-SNE, and the accuracy is evaluated using the metrics discussed above.
We can observe the results in Table 3.11 obtained by k means and PCA dimensionality reduction technique. The k-means algorithm converges if given sufficient time, but it highly depends on the initialization of centroids. So to resolve this we use k-means++ initialization scheme from scikit-learn. K-means++ initializes the centroids to be far away from each other, leading to good results than random initialization [10]. PCA-based method is used to reduce the dimensionality of the data and it helps in two-dimensional visualization with k-means clustering method. The k-means++ and its random initialization with PCA based method is evaluation results shows that the accuracy metrics for the data points belonging to same clusters is more. As the first five metrics which depends on the ground truth assignments (true labels) are all above 0.5 . The last metric silhouette coefficient where the ground truth labels are not considered and the model itself evaluated and the value is within 0 to 0.5, hence proving that the points belongs to the same clusters. The PCA reduced data visualization of the mushroom dataset is as shown in Figure 3.13.

Next, we implement another dimensionality reduction technique basic autoencoders which has a input layer with 117 dimensions, one hidden layer, encoder and decoder

same as discussed earlier. The input data X is splitted into training and testing data. The reduced input data X (low-dimensional) from autoencoders is given to k means clustering algorithm. The evaluation of autoencoders and k-means clustering performance can be seen in Table 3.12. As the ground truth is known for the first five metrics, we check the goodness of fit of the cluster labels to the ground truth. The first five metrics for the clustering performance are all above 0.5 proving that the data points well belongs to same clusters. The last metric Silhouette coefficient which is evaluated by the model itself without the labels gives values between 0 to 0.5, showing that the clusters are not highly dense. The visualization of mushroom dataset by basic autoencoders can be seen in Figure 3.14.

Finally, we compare the above two methods with the implementation of t-SNE using k means clustering on the mushroom dataset. The evaluation of the clustering is found for k means clustering method with t-SNE based algorithm as shown in Table 3.13.
The clustering evaluation for k-means with random initialization, k-means++ and t-SNE shows good results as the value of all the metrics are well between 0 and 1. The visualization of the mushroom dataset using t-SNE can be seen in Figure 3.15, it shows well separated clusters illustrating that the data points well belongs to the same cluster. Hence t-SNE with k-means clustering method performs better in comparison to autoencoders.

n_classes:2, n_samples:8124, n_features:117						
init	ARI	AMI	Homo	Compl	V-meas	Silhouette
k-means++	0.125	0.192	0.192	0.303	0.235	0.089
random	0.621	0.563	0.563	0.584	0.574	0.099
PCA-based	0.603	0.547	0.547	0.570	0.558	0.104

Table 3.11: Evaluation results of PCA and k-means clustering

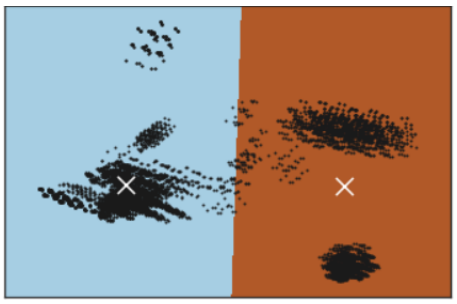

Figure 3.13: Visualization of mushroom dataset by PCA-reduced data and k-means clustering

n_classes:2, n_samples:8124, n_features:117						
init	ARI	AMI	Homo	Compl	V-meas	Silhouette
k-means++	0.229	0.284	0.284	0.371	0.322	0.100
random	0.606	0.537	0.537	0.556	0.547	0.107
autoencoder	0.603	0.547	0.547	0.570	0.558	0.087

Table 3.12: Evaluation results of autoencoders and k-means clustering

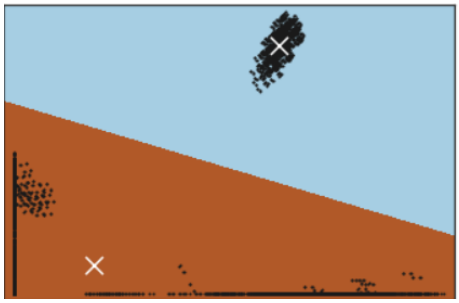

Figure 3.14: Visualization of mushroom dataset by autoencoders and k-means clustering

n_classes:2, n_samples:8124, n_features:117						
init	ARI	AMI	Homo	Compl	V-meas	Silhouette
k-means++	0.611	0.546	0.546	0.565	0.555	0.089
random	0.619	0.563	0.563	0.585	0.574	0.123
t-SNE	0.603	0.547	0.547	0.570	0.558	0.094

Table 3.13: Evaluation results of t-SNE and k-means clustering

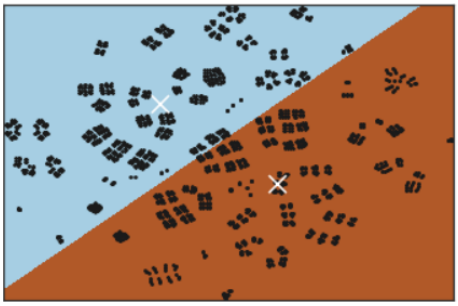

Figure 3.15: Visualization of mushroom dataset by t-SNE and k-means clustering

4 Conclusion

The experiments were conducted using the dimensionality reduction methods like autoencoders and t-SNE for visualization of breast cancer and mushroom datasets. The breast cancer dataset with 31 dimensions and the mushroom dataset with 117 dimensions is reduced to 2 dimensions for visualization using autoencoders and t-SNE.
Initially, We visualized the breast cancer dataset using basic and deep autoencoders. For classification, we used random forests a supervised learning algorithm. We also compared the basic autoencoders with adadelta and adam optimizers. The evaluation results of basic autoencoders with adam optimizers shows better performance in terms of metric precision, recall, and F1-measure.
Next, we built deep autoencoders with additional two hidden layers which helped in extracting more abstract features in comparison to basic autoencoders with 92 percent accuracy. We also visualized breast cancer dataset with PCA which gave us good accuracy, however, we just use PCA as a tool for visualization.
Further, we used another dimensionality reduction method t-SNE for visualizing the breast cancer dataset. The evaluation results of t-SNE is better than autoencoders with an accuracy of 96 percent. The perplexity being the main hyperparameter which gives different visualizations for its values ranging between 30 to 50.
We also implemented various cost functions for autoencoders to enhance the error method using mushrooms dataset. The performance and convergence line plots of deep autoencoders was smoother and better with binary cross-entropy in comparison to hinge loss and mean squared error.
Similarly, we visualized mushroom dataset using autoencoders and t-SNE with unsupervised k-means clustering method. The evaluation of clustering is based on metrics Homogeneity score (homo), Completeness score (compl), V measure (v-meas), Adjusted rand index (ARI), Adjusted mutual information (AMI) and Silhouette coefficient (silhouette). The t-SNE showed better visualization with well-separated clusters and evaluation accuracy which are well between 0 and 1. The cost function parameter perplexity helping in a smooth measure of the number of neighbors in the dataset. Also, the heavy-tailed distribution used in t-SNE helped to resolve the crowding problem in SNE. Thus, t-SNE outperforms the autoencoders in terms of evaluation results for classification and clustering along with better visualization.

5 References

[1] Yin, H., 2007. Nonlinear dimensionality reduction and data visualization: a review. International Journal of Automation and Computing, 4(3), pp.294-303.

[2] Xu, J.L., Xu, B.W., Zhang, W.F. and Cui, Z.F., 2008, July. Principal component analysis based feature selection for clustering. In 2008 international conference on machine learning and cybernetics (Vol. 1, pp. 460-465). IEEE.

[3] S. Haykin. Neural Networks. A Comprehensive Foundation (2nd Edition). Prentice Hall, Englewood Cliffs, NJ, USA, 2001.

[4] Kwak, N., 2008. Principal component analysis based on L1-norm maximization. IEEE transactions on pattern analysis and machine intelligence, 30(9), pp.1672-1680.

[5] Zhou, C. and Paffenroth, R.C., 2017, August. Anomaly detection with robust deep autoencoders. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 665-674).

[6] Windrim, L., Ramakrishnan, R., Melkumyan, A., Murphy, R.J. and Chlingaryan, A., 2019. Unsupervised Feature-Learning for Hyperspectral Data with Autoencoders. Remote Sensing, 11(7), p.864.

[7] Malhi, A. and Gao, R.X., 2004. PCA-based feature selection scheme for machine defect classification. IEEE Transactions on Instrumentation and Measurement, 53(6), pp.1517-1525.

[8] Maaten, L.V.D. and Hinton, G., 2008. Visualizing data using t-SNE. Journal of machine learning research, 9(Nov), pp.2579-2605.

[9] Patel, A.A., 2018. Hands-On Unsupervised Learning Using Python: How to Build Applied Machine Learning Solutions from Unlabeled Data. O'Reilly Media, Incorporated.

[10] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[11] C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998. Available online:https://archive.ics.uci.edu/ml/datasets/Mushroom

[12] Murphy, P., and Aha, D. (1994). UCI repository of machine learning databases.

Available online:https://archive.ics.uci.edu/ml/datasets/breast%20cancer

[13] “k-means++: The advantages of careful seeding” Arthur, David, and Sergei Vassilvitskii, Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics (2007).

[14] Vinh, N.X., Epps, J. and Bailey, J., 2009. Information theoretic measures for clusterings comparison New York.

[15] V-Measure: A conditional entropy-based external cluster evaluation measure Andrew Rosenberg and Julia Hirschberg, 2007.

[16] Rousseeuw, P.J., 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics, 20, pp.53-65.

[17] Vlachos, M., Domeniconi, C., Gunopulos, D., Kollios, G. and Koudas, N., 2002, July. Non-linear dimensionality reduction techniques for classification and visualization. In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 645-651).

[18] Gisbrecht, A., Schulz, A. and Hammer, B., 2015. Parametric nonlinear dimensionality reduction using kernel t-SNE. Neurocomputing, 147, pp.71-82.

[19] Brownlee, J., 2018. Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions. Machine Learning Mastery.

[20] Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep learning. MIT press.

[21] Marchi, E., Vesperini, F., Eyben, F., Squartini, S. and Schuller, B., 2015,April. A novel approach for automatic acoustic novelty detection using a de-noising autoencoder with bidirectional LSTM neural networks. In 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)(pp. 1996-2000). IEEE.

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, im August 2020