Spring 5-6-2016

Comparative Study of Dimension Reduction Approaches With Respect to Visualization in 3-Dimensional Space

Pooja Chenna

Comparative Study of Dimension Reduction Approaches with Respect to Visualization on 3-Dimensional Space

Master's Thesis

By

Pooja Chenna
MSCS Student
Department of Computer Science
Kennesaw State University, USA

Submitted in partial fulfillment of the
Requirements for the degree of
Master of Science in Computer Science

May 2016

Comparative Study of Dimension Reduction Approaches with Respect to Visualization on 3-Dimensional Space

This thesis is approved for recommendation to the Graduate Council.

[illegible]
Ying Xie
Thesis Advisor, Assistant Department Chair

[illegible]
Frank Tsui
Interim Department Chair, Committee Member

[illegible]
Dan Lo
MSCS Program Coordinator, Committee Member

[illegible]
Selena He
Assistant Professor, Committee Member

[illegible]
Yong Shi
Professor, Committee Member

DEDICATION

This thesis is dedicated to my family,

I thank everyone who supported me throughout my journey.

ACKNOWLEDGEMENTS

I would like to thank Dr. Ying Xie for his support, encouragement and motivation through this entire process.
I am very thankful for this experience.

I would also like to thank my thesis committee members, Dr. Shi, and Dr. He for their insightful comments and valuable suggestions.

This research paper is made possible through the help and support from everyone, including my professors, parents, my husband, family and friends.

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In the present big data era, there is a need to process large amounts of unlabeled data and find some patterns in the data to use it further. If data has many dimensions, it is very hard to get any insight of it. It is possible to convert high-dimensional data to low-dimensional data using different techniques, this dimension reduction is important and makes tasks such as classification, visualization, communication and storage much easier. The loss of information should be less while mapping data from high-dimensional space to low-dimensional space. Dimension reduction has been a significant problem in many fields as it needs to discard features that are unimportant and discover only the representations that are needed, hence it gathers our interest in this problem and basis of the research. We consider different techniques prevailing for dimension reduction like PCA (Principal Component Analysis), SVD (Singular Value Decomposition), DBN (Deep Belief Networks) and Stacked Auto-encoders. This thesis is intended to ultimately show which technique performs best for dimension reduction with the help of studied experiments.

TABLE OF CONTENTS

CHAPTER 1

Motivation, Problem Statement and Contribution

1.1 Background

With the advent of big data, lot of research is being done on how to deal with problems coming along. Big data refers to a large amount of structured and unstructured data usually in some petabytes and makes it tough to investigate more on the data. Thus problems like querying competently, analysing, visualization etc…arouse with big data. In this paper, we deal with the problem of dimension reduction and then visualization.

The challenges that big data is facing are given in terms of 4 V's namely "Volume", "Velocity", "Veracity" and "Variety" [1, 2]. In this context, dimension reduction comes into picture in dealing with one of the challenges of big data. High volume of data may result in multiple dimensions for data. It is really complex to analyze data when it has many dimensions, thus there should exist any of the dimension reduction techniques that help in reducing dimensions without losing the structure of data and it's meaning [3].

Traditional and deep-rooted methods of dimension reduction are done using factor analysis. Principal Component Analysis and Singular Value Decomposition come under this category. But these kind of techniques are not suitable for non-linear complex data. To efficiently deal with non-linear complex data many other dimension reduction techniques are proposed today but deep learning techniques prevail among them.

1.2 Motivation

When we consider dimension reduction techniques in particular, PCA and SVD are two examples of conventional methods whereas Deep Belief Networks and Stacked Auto-encoders are two examples of deep learning techniques that can easily handle non-linear and big data. Dimension Reduction is a significant problem in many real world studies because big data always has high dimensions and mapping the data from high dimensions to low dimensions is essential to increase the efficiency of data analysis and handling [4]. Most of the machine learning techniques till now exploited linear transformation using factorization or orthogonal projections in dimension reduction. These kind of techniques usually are not effective for non-linear feature transformations and but may be effective in solving many simple-data with limited constraints. Deep learning on the other hand has high representational power and deals with complex data.

Visualization and data compression are treated as the two important motivation subjects that incited interest in us about research in dimension reduction. Visualization can be performed in two ways, keeping all the dimensions or by reducing the dimensions. Heatmap, parallel co-ordinates, line graphs, Radviz, Polyviz etc.. are some of the visualization techniques. For example, parallel co-ordinates technique use ‘n’ parallel vertical lines if there are ‘n’ dimensions where the co-ordinates vary from minimum and maximum values for nth dimension. Now a poly line (represented as red and green lines in Figure 1) is drawn connecting all the ‘n’ parallel vertical lines or axes to represent a data point [5, 6].

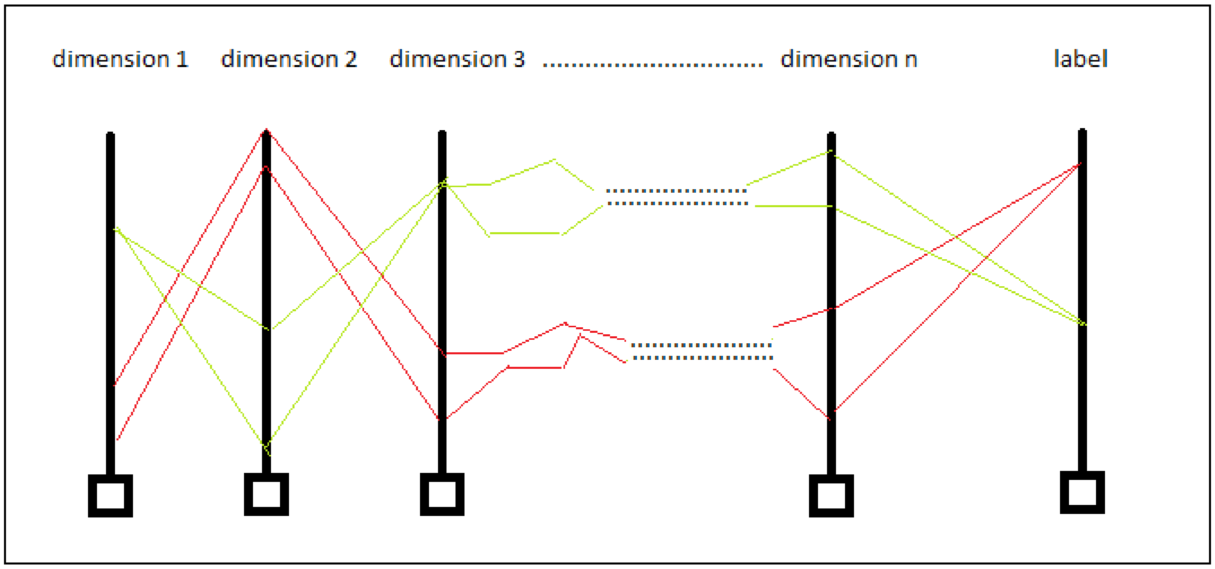


Figure 1: Parallel co-ordinate display for n-dimensional dataset

Figure 1 has parallel axes denoted by dimension 1, dimension 2, ……, dimension n. Green poly lines represent one type of class and red poly lines represent another type of class.

Similarly heat map uses a data grid or array of cells and color it based on the data points, line graphs use a continuous function in the form of a separate graph for each dimension. Representation and details about Radviz can be obtained from [7] and polyviz is just an extension for radviz with dimensions represented in the form of lines and not points to give more insight about data distribution [8]. But all these techniques where all the dimensions are kept do not help a lot in data exploration and analysis. It is impractical to visualize the data with high dimensions, hence dimension reduction has to be performed for an intuitive visualization. There are many ways to perform dimension reduction using PCA, MDS, Isomap, LLE, neural networks, SVD etc…We choose to compare PCA, SVD and deep learning techniques in this study.

Research work in [9] clearly states about the challenges faced by big data visualization like real-time scalability, perceptual scalability and interactive scalability. Summary of all these challenges include data being large, even with visualization it is difficult for a human to extract meaningful data, limited screen availability because of which everything cannot be seen, limitation of data size or storage to visualize, complex querying may freeze or crash the visualization systems.

Dimension reduction is the most common step used for data reduction and extracting information from big data. Few dimension reduction techniques have the ability to remove the correlation among the data variables and few others have data divided among the clusters which simplifies the process of data analysis. Reducing the dimensions to 2 or 3 helps in improved visualization. For example, scatter plots shown in Figure 2.

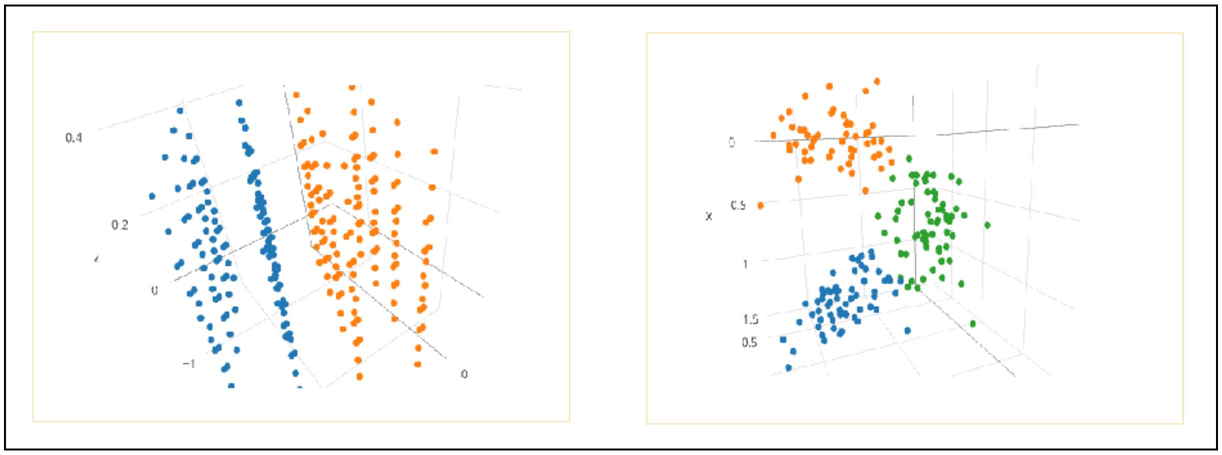


Figure 2: Two scatter plots of random datasets in 3-D space

A scatterplot is a point projection of the data into 2D or 3D space and one of the most used visualization methods [8]. Visualization in 3D space shown in Figure 2 is so clear and it is further easy to predict about any new data in these type of plots. This can be easily explained using an example scenario using breast cancer dataset.

Breast cancer dataset

Table 1: Attribute information for breast cancer dataset

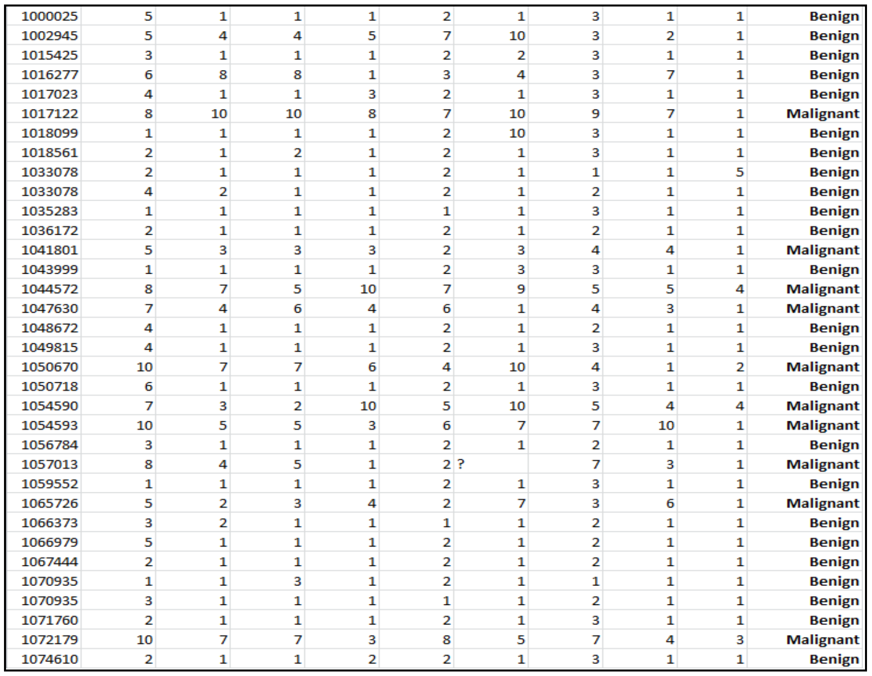
Number of Attributes : 9
Number of classes : 2
Number of Samples : 699

Attribute Information:

1. Clump Thickness
2. Uniformity of Cell Size
3. Uniformity of Cell Shape
4. Marginal Adhesion
5. Single Epithelial Cell Size
6. Bare Nuclei
7. Bland Chromatin
8. Normal Nucleoli
9. Mitoses

Benign or Malignant – Classes

Table 1 gives the details about breast cancer dataset such as number of dimensions and number of classes it has.



1000025	5	1	1	1	2	1	3	1	1	**Benign**
1002945	5	4	4	5	7	10	3	2	1	**Benign**
1015425	3	1	1	1	2	2	3	1	1	**Benign**
1016277	6	8	8	1	3	4	3	7	1	**Benign**
1017023	4	1	1	3	2	1	3	1	1	**Benign**
1017122	8	10	10	8	7	10	9	7	1	**Malignant**
1018099	1	1	1	1	2	10	3	1	1	**Benign**
1018561	2	1	2	1	2	1	3	1	1	**Benign**
1033078	2	1	1	1	2	1	1	1	5	**Benign**
1033078	4	2	1	1	2	1	2	1	1	**Benign**
1035283	1	1	1	1	1	1	3	1	1	**Benign**
1036172	2	1	1	1	2	1	2	1	1	**Benign**
1041801	5	3	3	3	2	3	4	4	1	**Malignant**
1043999	1	1	1	1	2	3	3	1	1	**Benign**
1044572	8	7	5	10	7	9	5	5	4	**Malignant**
1047630	7	4	6	4	6	1	4	3	1	**Malignant**
1048672	4	1	1	1	2	1	2	1	1	**Benign**
1049815	4	1	1	1	2	1	3	1	1	**Benign**
1050670	10	7	7	6	4	10	4	1	2	**Malignant**
1050718	6	1	1	1	2	1	3	1	1	**Benign**
1054590	7	3	2	10	5	10	5	4	4	**Malignant**
1054593	10	5	5	3	6	7	7	10	1	**Malignant**
1056784	3	1	1	1	2	1	2	1	1	**Benign**
1057013	8	4	5	1	2	?	7	3	1	**Malignant**
1059552	1	1	1	1	2	1	3	1	1	**Benign**
1065726	5	2	3	4	2	7	3	6	1	**Malignant**
1066373	3	2	1	1	1	1	2	1	1	**Benign**
1066979	5	1	1	1	2	1	2	1	1	**Benign**
1067444	2	1	1	1	2	1	2	1	1	**Benign**
1070935	1	1	3	1	2	1	1	1	1	**Benign**
1070935	3	1	1	1	1	1	2	1	1	**Benign**
1071760	2	1	1	1	2	1	3	1	1	**Benign**
1072179	10	7	7	3	8	5	7	4	3	**Malignant**
1074610	2	1	1	2	2	1	3	1	1	**Benign**

Figure 3: Snapshot for breast cancer dataset

Figure 3 shows initial few samples from breast cancer dataset, it gives us an idea on how data looks like. Classes are labeled as 0 or 1 instead of malignant and benign to serve as input for dimension reduction technique. The breast cancer data with 9 dimensions is chosen to be reduced to 3 dimensions for better visualization with the help of Principal Component Analysis, one of the dimension reduction techniques.

Two classes malignant and benign for breast cancer dataset are shown as blue circles and red squares in Figure 4. It is clear from Figure 4 that the two classes are easily differentiable and separable. That is the perfect advantage of visualization. Now Figure 5 talks about new data points whose class is unknown and those are shown in a different color (black).

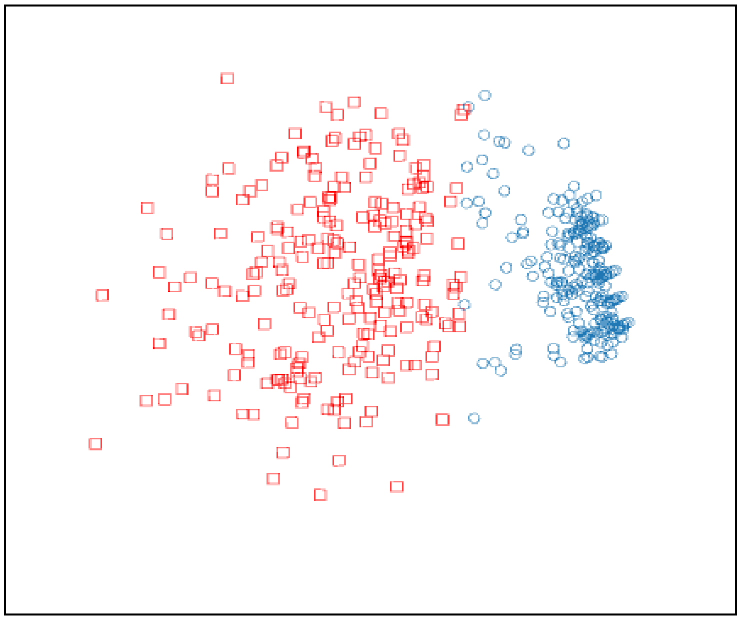


Figure 4: 3D scatter plot for breast cancer dataset with PCA

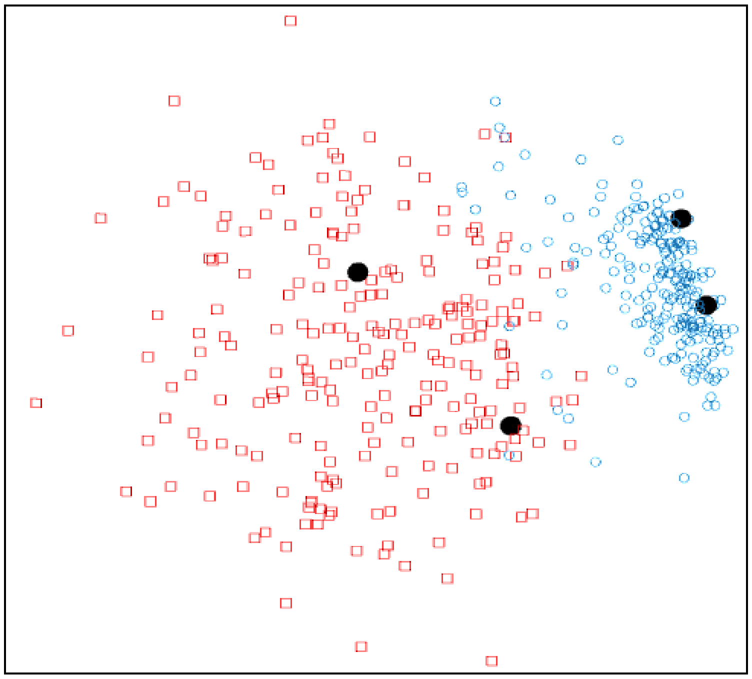


Figure 5: New data with unknown classes (in black circles) in breast cancer dataset

Class labels for four samples from the original reduced dataset are made unknown for the purpose of making visual decision. The resultant dataset plot is shown in Figure 5 where the four black circles represent unknown class samples. But from the visualization in Figure 5, class can be easily predicted based on examining the cluster to which this unknown class sample belongs to. This simple illustration with the help of Figure 6 makes us understand the value and benefits of visualization.

At the end, in order for an effective and intuitive visualization, data dimension reduction is considered to be an important step. At the same time, quality of dimension reduction and quality of visualization has to be ensured for efficient data analysis. So, visualization of high dimensional data preserving the original data's intrinsic structure is the motive behind this research.

1.3 Problem Statement

Preserving the structure of data after dimension reduction plays an important role when dealing with big data. Though there are many techniques for reducing dimensions, it is necessary to check that data is reduced with minimum loss of information. If there is much loss in structure or meaning of data, the original motive of analyzing the data with lower dimensions may not be achieved properly. This work is an effort to research and propose a new approach to address this kind of problem.

During literature search, we identified a list of common limitations among past research efforts. These limitations are summarized as follows:

- Lack of thorough research that compare traditional approaches with deep learning approaches in terms of dimension reduction. In particular, most efforts are effective to compare linear techniques with non-linear techniques for dimension reduction. However, they are complete different type of research and may not be related to deep learning.
- No evidence is provided in the research about maintaining the structure and originality of data after reducing it from higher dimensions to lower dimensions. Also very few strategies are provided in order to achieve the same. In particular, there is no effort of formally specifying the methods to show that the data structure is preserved after dimension reduction.

Given the obtained literature search results, we define the problem statement for this thesis as follows:

This research work addresses common limitations found in past research efforts by executing an in-depth study of dimension reduction, provides visualization for high dimensional datasets while keeping the original data structure, in order to evaluate, an approach to check the information loss for original data after reducing dimensions is designed, and the proposed approach makes the comparison easier.

1.4 Research Methodology

The research methodology comprised of an intensive literature review of different articles consisting of information on big data, machine learning, deep learning, linear and non-linear dimension reduction, deep learning techniques. The research methodology involves the following activities:

1) Conduct literature search on existing comparison reviews for dimension reduction and their performance.
2) Study and analyze collected information to understand how different techniques perform on data after application and reducing dimensions.
3) Develop deep belief networks, a technique which is proposed for comparison but non-existing in HPCC deep learning module.
4) Propose an approach to check whether the loss of information is minimum for the data after dimension reduction.
5) Evaluate the proposed approach against different dimension reduction techniques for comparison.

1.5 Contribution

The objective of this thesis is to conduct an in-depth comparison of various types of dimension reduction techniques like PCA, SVD, Stacked Auto Encoders, Deep Belief Networks; study of existing methodologies for dimension reduction; develop an approach to check the information loss after reducing the dimensions; evaluate the proposed approach.

The work addresses the stated problem statement by performing the following tasks:

1) Conduct Literature Search and Develop Deep Belief Networks

   i. Conduct literature search on existing deep belief network implementations in other platforms and check for their accuracy. It has to be replicated in HPCC.

   ii. Conduct literature search on different linear and non-linear data dimension reduction techniques. An approach has to be determined for checking whether the structure of data is preserved.

   iii. The author will compose a survey of compiled papers related to this topic and document the findings.

2) Develop an Algorithm and Approach

   i. The algorithm for Deep Belief Networks is implemented and tested for accuracy.

   ii. The developed algorithm is tested with the given input datasets to reduce their dimensions and visualized in a 3D space to analyze further.

   iii. An approach is determined for checking the structure of data after reducing the dimensions.

3) Get Datasets

   i. Suitable datasets are gathered to perform experiments.

   ii. 5 datasets with more than 30 dimensions to datasets having 6 dimensions are considered for the research. Each dataset is different from its perspective. Basically they all have different characteristics, hence considered for testing.

4) Comparison Study

   i. Perform simulations for all the techniques and use the unsupervised clustering, entropy determination and visualization approach for better comparison.

5) Dissemination of Results

   i. Disseminate the results for ease of access and understanding. The source code will be available upon request, but this is at the discretion of the author.

   ii. Prepare and submit one or more papers for publication at related venues

In the next Chapter, we present literature review for all the dimension reduction techniques that are used for comparative study.

CHAPTER 2

Literature Review

2.1 Overview

Many research works have been done in the past to compare dimension reduction techniques [10, 11, 12, 13, 14] and other works are presented in related work section 3.2. This Chapter presents a literature review of different dimension reduction techniques and their recent applications.

2.2 Dimension reduction techniques

Upon literature review, we consider two traditional approaches and two deep learning approaches for comparison. We list below all the four approaches used for comparison in this paper.

SVD (Singular Value Decomposition):

Singular Value Decomposition is a matrix factorization method. Formally, if there is a dataset with m*n dimensions, there exists a factorization called singular value decomposition of M, of the form

$$M = U\Sigma V^*$$

where U is a unitary matrix, Σ is a diagonal matrix with non-negative real numbers on the diagonal and V is a unitary matrix. The diagonal entries of Σ are known as singular values of M.

“Singular value decomposition components of a matrix U, Σ and V can be multiplied together to recreate the original matrix exactly. However, if only a subset of rows and columns of matrices U, Σ, and V are used, then those lower-order matrices U, Σ, and V provide the best approximation of the original matrix in the least square error sense. Because of that, SVD can be seen as a method for transforming correlated variables represented by columns of the original matrix into a set of uncorrelated variables that better expose relationships that exist among the original data items. SVD can also be used as a method for identifying and ordering the dimensions along which data points exhibit the most variation.“[15]

PCA (Principal Component Analysis):

“Principal Component Analysis is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called ‘principal components’. The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, it accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible, under the constraint that it is orthogonal (meaning uncorrelated with) to the preceding components.” [15]

DBN (Deep Belief Networks):

Deep Belief Networks is a probabilistic generative model composed of multiple layers of stochastic, hidden variables [16]. Before knowing about deep belief networks, it is important to know about Restricted Boltzmann Machines (RBMs) because RBMs are stacked to form so called Deep Belief Network. In 2006, Hinton showed that RBMs can be stacked and trained in a greedy manner to form DBNs. DBNs are graphical models which learn to extract a deep hierarchical representation of the training data.

RBMs use an algorithm called “Contrastive Divergence” instead of traditional back propagation to learn and prepare a model. The Contrastive Divergence algorithm works in two phases namely positive and negative phases. In positive phase, the input vector ‘v’ is clamped to the input layer and is propagated to hidden layer in a similar manner to feed forward neural networks and obtain a result ‘h’. In negative phase, the result ‘h’ from positive phase is propagated back to the visible layer, obtains a result v’ and the new result is again propagated to hidden layer with activation result h’.

After the positive and negative phases, weight is updated as:

$$w(t+1) = w(t) + \alpha(vhT - v'h'T)$$

where α is the learning rate and w, v, v’, h, h’ are vectors.

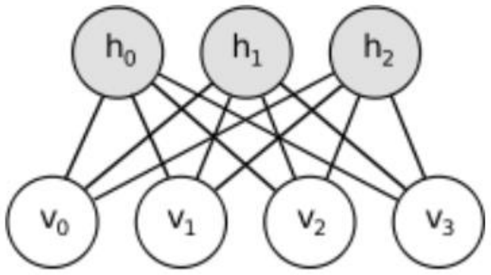


Figure 6*:* Restricted Boltzmann Machine –Image is copied from [16]

The intuition behind the algorithm is that the positive phase reflects the network's internal representation of the real world data. Meanwhile, the negative phase represents an attempt to recreate the data based on this internal representation. The main goal is for the generated data to be as close as possible to the real world and this is reflected in the weight update formula. In other words, the net has some perception of how the input data can be represented, so it tries to reproduce the data based on this perception. If its reproduction isn't close enough to reality, it makes an adjustment and tries again [17].

There are different representative works using Restricted Boltzmann Machines like Deep Belief Networks, Deep Boltzmann Machines, Deep Energy Models [18]. One of the works "Deep Belief Networks" is further discussed.

Figure 7 is a representation for deep belief network. These deep belief networks are often quite powerful producing impressive results.

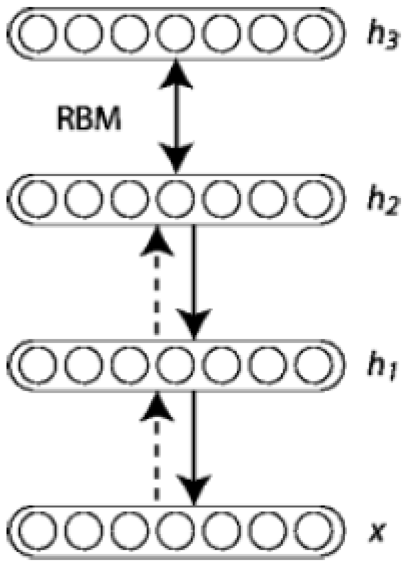


Figure 7: Deep Belief Network – Image is copied from [16]

DBNs model the joint distribution between observed vector ‘x’ and the ‘l’ hidden layers h^k as follows:

$$P(x, h^1, \ldots, h^\ell) = \left(\prod_{k=0}^{\ell-2} P(h^k | h^{k+1}) \right) P(h^{\ell-1}, h^\ell)$$

where $x = h^0$, $P(h^{k-1}|h^k)$ is a conditional distribution for the visible units conditioned on the hidden units of the RBM at level k, and $P(h^{l-1}|h^l)$ is the visible-hidden joint distribution in the top-level RBM.

Despite these impressive characteristics of deep belief networks that suits for reducing dimensions, we also mentioned few papers in the related work – section 3.2 that explicitly discussed deep belief networks for dimension reduction.

Stacked Auto-encoders:

Stacked auto-encoders are best used for unsupervised learning as it is good in capturing hierarchical groups, that is primary layers of the network learns higher level features and as we go deeper in network, it tries to learn lower level features in deep learning that replaced the learning techniques used in conventional neural networks.

Stacked Auto-encoders as the name suggests is a stack of auto-encoders. They are also referred to as Stacked Auto-Associators as they try to associate the output with the input and try to find intermediate representations [19]. Traditionally, in stacked auto-encoders output from one auto-encoder is treated as input for the next auto-encoder and this process repeats till all the individual auto-encoders in the network are pre-trained [20]. The result at the output layer is with reduced dimensions. There are different variations in auto-encoder like Sparse Autoencoder, Denoising Autoencoder, Contractive Autoencoder [18]. For dimension reduction, we do not consider the attribute sparsity because the original dimensional space is reduced but not expanded.

Table 2: Comparison between RBMs and autoencoders

Properties	**RBMs**	**Autoencoders**
Generalization	Yes	Yes
Unsupervised Learning	Yes	Yes
Feature Learning	Yes	Yes
Real-time training	No	Yes
Real-time prediction	Yes	Yes
Biological Understanding	No	No
Theoretical Justification	Yes	Yes
Invariance	No	No
Small training set	Yes	Yes

Table 2 is the comparison study between only two deep learning techniques restricted boltzmann machines and autoencoders. It is extracted from [18] where the comparison is performed among different other deep learning techniques also. From the comparison it is clear that deep learning techniques do not depend on invariance at all whereas the traditional techniques depend on variance and orthogonal transformations.

According to [21], research on dimension reduction has taken many sides like it can be done with projections or making use of neural networks or similar data or fractality. Our paper discusses dimension reduction from the sides of projections and neural networks. It is also stated in [21] that linear techniques like PCA have more time complexity and space complexity of o(m^2), so they have designed an optimized RBM approach with dynamic hidden layers to show the better results on MNIST [18] database. Our RBM approach uses fixed number of hidden layers but achieves good results after trying multiple combinations but work accomplished in [21] is optimized by using dynamic hidden layers.

Apart from dimension reduction, deep learning techniques can be used for classification purposes also which is illustrated in [19, 22]. In [22], abstraction feature of DBN is combined with back propagation strategy for classification. A slight variation of auto-encoders is used to solve the classification problem with minimal errors in [19] and performance gap is reduced when compared to DBN.

The advantages of deep learning techniques over traditional techniques are:

- It uses unsupervised training which eliminates the need of labels for training.
- Local optima can be prevented.
- Data can be separated more easily.
- Meaningful representations can be made.

Due to the growing research on non-linear dimension techniques, which necessarily need not be deep learning techniques but may be variations of PCA like Kernel PCA and others like LLE, HLLE etc…one may have the intuition that non-linear techniques are more preferred to linear techniques. But through research conducted in [23], it may be clear that it is not always non-linear techniques that prevail. For datasets with a lot of noise and lot many outliers, non-linear techniques are not best suitable. Table 3 gives categorization of all the dimension reduction techniques that makes linear, non-linear, traditional and deep learning terminologies clear.

Table 3: Categorization of dimension reduction techniques

Reduction Technique	**Type of suitable data**	**Type of technique**
PCA	Linear	Traditional
SVD	Linear	Traditional
Deep Belief Networks	Linear and Non-linear	Deep Learning
Stacked Autoencoders	Linear and Non-linear	Deep Learning

For deep learning techniques, if there is one hidden layer and linear with certain nodes, then the projection is similar to PCA. If the hidden layers are non-linear, then there can be different kinds of abstraction which lead to better results [4].

Linearity and non-linearity mostly differ in the cost function used in respective approaches. In traditional PCA and SVD approaches, it’s just the matrix factorization involved which is linear. But in other non-linear techniques and deep learning techniques complex functions are used to deal with non-linear data.

Though it is learnt that there are many techniques for performing the task of dimension reduction, it is important to realize how to reduce the dimensions. Dimension reduction involves two important steps namely variable selection and feature extraction [24]. All the approaches discussed in this paper PCA, SVD, deep belief networks and stacked auto-encoders are feature extracting methods. They concentrate on finding the best features from given set of high dimensional space which represent the original data. It can be found based on variance if it is traditional approach or based on abstraction property if it is deep learning technique.

In the next Chapter, we introduce approach on how to check whether the structure of original data after dimension reduction is preserved.

CHAPTER 3

K-means Clustering based Approach, Dimension Reduction, Visualization and Entropy Determination

3.1 Overview

This Chapter introduces the works done by different authors and strategies used for comparison in Section 3.2. Next we introduce the approach on entropy determination after finding the good ‘k’ value, reducing the dimensions and then performing the clustering of reduced data to check the entropy of which technique is low. However, the goal is to visualize high dimensional data in a 3D space intuitively while keeping the original data structure as much as possible. Thus proposed approach helps in realizing our goal. Benefits of dimension reduction and visualization are given with an example in chapter 1 under motivation. The design details and strategy used for comparison in this paper is clearly given in Section 3.3. Section 3.4 is to list out the benefits of proposed approach when compared to existing approaches and section 3.5 explains how the reduction is performed with each of the dimension reduction techniques.

3.2 Related Work

In [3], a comprehensive comparative study of 12 linear and non-linear techniques are used for dimension reduction. Testing is performed on artificial and natural datasets. Evaluation criteria used is based on generalization errors in classification tasks. K-nearest neighbor classifier is employed because of its high variance. It is believed by the authors that high variance helps in judging the structure of the data. They also chose generalization errors rather than reconstruction errors because no conclusion can be drawn when the reconstruction error is high, it may not mean that dimension reduction did not perform well. Finally based on the experiments it is concluded in the paper that in spite of high variance exhibited by non-linear techniques, they are not much better when compared to traditional PCA for many datasets that do not rely on local properties.

Some relevant works on dimension reduction [4, 10, 11 and 25], some related works on visualization [13, 26, 27 and 28] and many others are shown in Table 4. These works use dimension reduction with different techniques. Deep Belief Networks and PCA comparison for dimension reduction with evaluation criteria being sum of squared errors difference between original one and reconstructed one is provided in [4]. A new non-linear algorithm is proposed based on eigen-value face decomposition in [10] and is compared with PCA. A non-linear generalization for PCA which uses a encoder and decoder network is used in [11], it is a neural network based dimension reduction and root mean squared error is the evaluator. Stacked auto encoders are used for dimension reduction in [25] where it is concluded that, they are not only good at reducing but also in finding repeated structures. Again a new non-linear algorithm named "Distinguishing Variance Embedding (DVE)" is designed combining the concepts of maximum variance unfolding and Laplacian Eigenmaps in [29]. In this paper, the criteria used for preserving data structure in Laplacian Eigenmaps like sum constraint is improved with strict local preserving constraints which is achieved by maximizing global variance for constraints obtained in Laplacian Eigenmaps. In the illustration, 2-dimensional and 3-dimensional embeddings show that the original local neighborhood is preserved and also in images instance, some features are clearly distinguished.

In few of the works where visualization itself is used for evaluation, it is difficult to verify the quality of visualization. A new metric based on pairwise correlation of the geodesic distance is proposed in [23]. In order to prove that the proposed metric performs better on many dimension reduction techniques, they compared it with several other metrics like Euclidean distance, spearman distance etc…

Our work is mainly focused on comparing the traditional approaches with deep learning approaches for dimension reduction. We focus on visualization of every dataset after reduction and try to validate our analysis based on visualization with the help of entropy.

In contrast to earlier work, our proposed approach gives a better way to visualize and check the loss of information using entropy determination. Entropy is used in machine learning algorithms such as decision trees to find the information gain, it is one of the powerful criteria to check on loss of information [30].

Table 4: COMPARISON OF RELATED WORK

Work	**Reduction Techniques**	**Datasets**	**Evaluation Criteria**
Noulas *et al.* [4]	Deep Belief Network	AR Face Database [17]	Difference between sum of squared errors of original and reconstructed one
Gering [10]	Eigen Value Face Decomposition	Random Faces	Reconstruction Accuracy
Teli [11]	Neural Network (Encoder – Decoder)	MNIST [18], USPS, Olivetti Face Dataset [19]	Root Mean Square Error
Maaten *et al.* [12]	PCA, Isomap, MVU, Kernel PCA, Diffusion Maps, Autoencoders, LLE, Laplacian Eigen maps, Hessian LLE, LTSA, LLC, Manifold Charting	Artificial and Natural Datasets	K-nearest neighbor generalization error
Tsai [13]	PCA, MDS, LSA, Isomap, LLE, HLLE, LTSA	Blog Entries	Visualization and finding outliers
Claveria *et.al* .[14]	CATPCA (categorical PCA), MDS	Tourist Destinations (categorical data)	Trending tourism is analysed in top 10 world destinations which is shown graphically
Wang *et al.* [25]	Stacked Auto-encoders	Synthesized Data, MNIST [18], Olivetti Face Dataset [19]	Visualization
Venna *et.al.* [26]	PCA, MDS, LLE, Laplacian Eigenmap, HLLE, Isomap, CCA, CDA, maximum variance unfolding, LMVU, local MDS	Plain s-curve dataset, noisy s-curve dataset, mouse gene expression, gene expression compendium, sea-water temperature time series	Neighborhood Retrieval Visualizer
Najim *et.al.* [27]	PCA, CCA, CDA, Trustworthy Stochastic Proximity Embedding and 17 other linear and non-linear methods	Synthetic data (curved cylinder), Tensor colored image dataset	Quality of visualization using residual variance, correlation function and local continuity
Dzwinel *et.al.* [28]	nr-MDS (variant of MDS)	MNIST [18], Reuters	Interactive Visualization
Wang *et al.* [29]	Distinguishing Variance Embedding	Synthetic Data (helix, Swiss roll, Punctured Sphere, Twin Peaks, Gaussian) and	Strict local distance-preserving constraints

Work	Reduction Techniques	Datasets	Evaluation Criteria
		Image Datasets (COLI-20 Database [28], MNIST [18])	
Our approach	PCA, SVD, Deep Belief Networks, Stacked Auto encoders	KDD 1999 Cup Data, Breast Cancer, Cover Type	K-means clustering approach, finding entropy and visualization.

3.3 Unsupervised Approach for Evaluating Performance of Dimension Reduction With Respect To Visualization

Ideally, we can take the original dataset with any number of dimensions, use any of the dimension reduction techniques to make it low dimensional data and then visualize it. But in order to visualize high-dimensional data in an intuitive way, we need to reduce the dimensions to three but at the same time ensure that whether reduced data is having same structure as the original data, for which we use entropy to evaluate. This approach involves unsupervised k-means approach to group the data and evaluation using entropy determination, then comparison is performed. A step wise approach is shown in the Table 5.

Table 5: Step-by-Step procedure for reduction and evaluation

Step 1. *Examine the data, preprocess the data if required*
Step 2. *Run k-means on it with wide range of k-values.*
Step 3. *Determine the suitable k-value for the dataset.*
Step 4. *Discard the original labels and assign cluster numbers as labels for all the samples.*
Step 5. *Perform dimension reduction for original data.*
Step 6. *Visualize the data in 3D space.*
Step 7. *Using the new cluster labels, find the entropy for the reduced data to evaluate and to compare.*

The value of entropy in Step 7 of Table 5 talks about the amount of loss of information, if the entropy is low then the loss is less. Thus entropy can be used as a measure to evaluate the quality of clustering [31] and also dimension reduction.

Our idea is to find the entropy values of reduced dataset by repeating the procedure explained in Table 5 for different dimension reduction techniques and then compare the entropy score. The lowest entropy score recorded for the dataset with the reduction technique used is chosen as the best approach for dimension reduction. It may again vary on type of data, sparsity, relations among data etc…This evaluation and result discussion is clearly stated with the help of experimental study in chapter 4 and detailed step-by-step explanation is provided in this section.

Step 1. Examine the data, preprocess the data if required.

It is not possible to have real time data within a specific range. Real time data usually falls under a large range and also there may be possibility of categorical values. In order not to give weightage for any specific attribute while using classification or clustering algorithms or in neural networks, preprocessing the data acts as an important step. Preprocessing may involve steps like normalization or standardization. Table 6 explains major differences between normalization and standardization.

Table 6: Difference between normalization and standardization

Normalization	**Standardization**
Used when maximum and minimum values of the dataset are known.	Used when maximum and minimum values of the dataset are unknown.
Standard deviation and variance are not involved.	Standard deviation and variance are involved.
Data after normalization is bounded within a range [32].	Data after standardization may not be completely bounded.

We use normalization for preprocessing the data for all the studied experiments since keeping data in a specific range is very important for all the algorithms used in comparative study. Normalization is specifically important before clustering because if there is a large variation among the values of attributes, one attribute may dominate over the other, hence the motive is to balance all the attributes without giving weightage to any attribute in specific [40]. It is completely user's choice to decide on which kind of normalization technique or rule has to be applied for specific dataset [33].

The kind of normalization chosen for this study is called min-max normalization. Using min-max normalization ensures data is in the range of 0 to 1. There are two min-max normalization forms again which are stated as in Table 7.

Table 7: Two forms of min-max normalization

Min-max normalization form 1 [32]	$(X-X_{min})/(X_{max}-X_{min})$
Min-max normalization form 2 [34]	$[(X-X_{min})/(X_{max}-X_{min})]*(newX_{max}-newX_{min})+newX_{min}$

In Table 7,

'X' refers to the data point

Xmax refers to the maximum value of X

Xmin refers to the minimum value of X

newXmax refers to maximum value of the interval range in which X has to be

newXmin refers to minimum value of the interval range in which X has to be

Techniques used for normalization will transform the original data but it should be confirmed that no noise is introduced in the original data [34]. Data transformation is usually linear in normalization approaches. More about normalization forms used for different datasets is presented in chapter 4.

Step 2. Run k-means on it with wide range of k-values.

The concept of k-means can be simply stated as "Use the data to move the centers" and "Use the centers to move the data" [35]. K-means is used for data that do not have labels and is a kind of unsupervised approach. It clusters the data based on the algorithm. 'k' in k-means refers to the number of clusters. Algorithm taken from [36] shown in Figure 8 clearly explains the basic concept of k-means clustering approach.

```
Algorithm 1: K-means($D$,$k$)
1 Let $i$=Float.MAXVALUE; $j$=1
2 Choose $k$ centers from $D$, let $C^{(0)} = c_1^{(j)}, c_2^{(j)}, ..., c_k^{(j)}$
3 while $i > itr$ do
4     form $k$ clusters by assigning each points in $X$ to its nearest center
5     find new centers of the $k$ clusters $c_1^{(++j)}, c_2^{(++j)}, ..., c_k^{(++j)}$
6     $i \leftarrow \sum_{m=0}^{k} ||c_m^j - c_m^{j-1}||^2$
7 output $C^{(j)}$
```

Figure 8: Algorithm for k-means [36]

The process of k-means is performed through a set of scala statements for the purpose of this study as scala has better options for machine learning library. Scala statements used for running k-means is shown in Table 8.

Table 8: Scala Statements for running k-means

```
def distance(a: Vector, b: Vector) =
    math.sqrt(a.toArray.zip(b.toArray).
      map(p=>p._1-p._2).map(d=>d*d).sum)

def distToCentroid(datum: Vector, model: KMeansModel) = {
    val cluster = model.predict(datum)
    val centroid = model.clusterCenters(cluster)
    distance(centroid, datum)
}


import org.apache.spark.rdd._
import org.apache.spark.mllib.clustering._

def clusteringScore(data: RDD[Vector], k:Int) = {
   val kmeans = new KMeans()
   kmeans.setK(k)
   kmeans.setRuns(10)
   kmeans.setEpsilon(1.0e-6)
   val model = kmeans.run(data)
   data.map(datum=>distToCentroid(datum, model)).mean()
}

(1 to 12 by 1).map(k=> (k, clusteringScore(data, k))).foreach(println)
```

In Table 8,
distance – function to calculate distance between two points
distToCentroid – function to calculate distance between data point and centroid
clusteringScore – function that helps in choosing the best ‘k’ value based on the clustering score. It sets ‘k’ value, sets the epsilon value that controls the movement of centroid in a cluster and also runs kmeans for 10 times to get a model.
data – input dataset in form of Vector.

Last statement in Table 8 prints the clustering score for a wide range of values specified based on step size (in this example, series goes like 1, 2, 3, 4,….., 11, 12 as step size is 1 and range specified is 1 to 12).

Figure 9 gives an idea of output for last scala statement (clustering score for wide range of ‘k’ values) in Table 8.

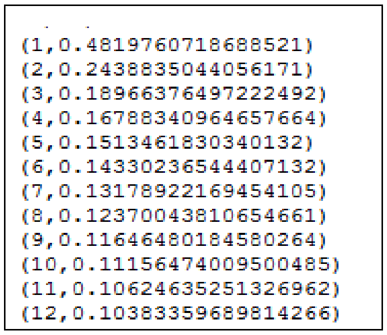

```
(1,0.4819760718688521)
(2,0.2438835044056171)
(3,0.18966376497222492)
(4,0.16788340964657664)
(5,0.1513461830340132)
(6,0.14330236544407132)
(7,0.13178922169454105)
(8,0.12370043810654661)
(9,0.11646480184580264)
(10,0.11156474009500485)
(11,0.10624635251326962)
(12,0.10383359689814266)
```

Figure 9: Clustering Score for random dataset

Now since we have clustering score for wide range of ‘k’, next step is to choose best ‘k’. It is very important to choose the best ‘k’ as it decides the best clustering.

Step 3. Determine the suitable k-value for the dataset.
One basic idea to choose the best ‘k’ is by observing the sharp change in clustering score. An easy way to determine the elbow or the sharp decrease is by plotting a graph. The elbow technique of choosing ‘k’ is taken from [31]. Figure 10 is the graph plotted for the data in Figure 9.

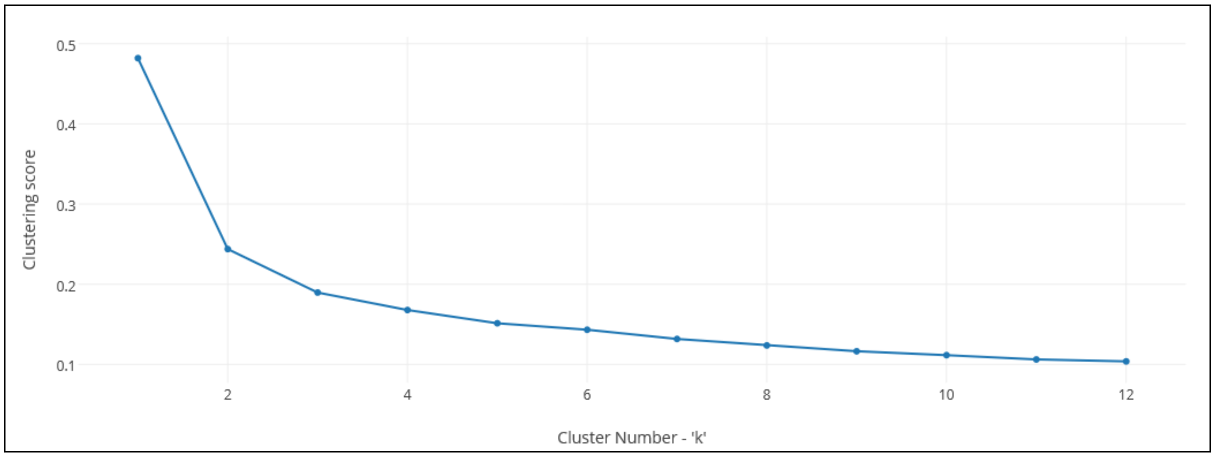


Figure 10: Graph for clustering score and wide range of k

From the Figure 10, a sharp decrease is observed at k-value ‘2’. So the best k-value is chosen to be ‘2’ for some random dataset that has clustering scores for wide range of ‘k’ as shown in Figure 9.

Step 4. Discard the original labels and assign cluster numbers as labels for all the samples.
After choosing the best ‘k’ value, k-means algorithm has to be run again on the original dataset with best ‘k’ chosen and same parameters that are used for choosing ‘k’.

Again a set of scala statements are used to run the k-means algorithm on the original dataset and predict the new clusters for the given data. Table 9 gives a piece of scala code to run k-means algorithm individually.

Table 9: Scala statements to run k-means with best chosen ‘k’

```
import org.apache.spark.mllib.clustering._

val kmeans = new KMeans()
kmeans.setK(2)
kmeans.setRuns(10)
kmeans.setEpsilon(1.0e-6)

val model = kmeans.run(data)
val sample = data.map(datum =>
model.predict(datum) + "," + datum.toArray.mkString(",")
)
```

```
sample.saveAsTextFile("/user/pchenna/clustered_data”)
```

Code in Table 9 is not much different from code represented by using clusteringScore function in Table 8. Earlier, function is called multiple times for different values of ‘k’ but here k-means run for the best chosen ‘k’ (in this case k=2 as per step 3).

Once k-means is run with all the parameters set, model obtained is used to predict the cluster number for each sample in the original dataset. At this point, we believe that good clustering is performed with best determined ‘k’. In other words, all the homogenous samples are grouped together. So, we choose to deal with cluster numbers as labels rather than the original labels in further process. Hence as a last thing in Step 4, original labels are discarded and cluster numbers (cluster to which sample belongs to) are assigned as labels to all the samples in the original data.

“model.predict(datum)” in Table 9 predicts the cluster to which the sample belongs to, based on k-means model. After prediction, data with new cluster labels is saved in a folder called “clustered_data” from where it can be retrieved.

Step 5. Perform dimension reduction for original data.

In this step, original data dimensions are reduced to 3 using different reduction techniques. More about the dimension reduction techniques used and obtaining the data with reduced dimensions using each of the techniques is discussed in section 3.5.

Step 6. Visualize the data in 3D space.

In order to compare the different dimension reduction techniques, we choose visualization as a tool. Reduced dimensional data with each of the techniques is visualized in 3D space. Even after data reduction by dimension, there should not be much loss of information and structure. Thus visualization gives a better idea to the user about the intrinsic structure of data after reducing it to 3 dimensions. Benefits of visualization are already discussed in chapter 1.

Step 7. Using the new cluster labels, find the entropy for the reduced data to evaluate and to compare

Though visualization gives us better idea about the structure of the data when realized in 3D space, yet it is sometimes difficult to compare different visualizations. There has to be a way using which closer visualization for data can be differentiated. We have found that entropy as a measure which does its best job in this aspect. Further details in this Step speak about the calculation of entropy and how entropy is used to compare the quality of dimension reduction.

Table 10: Dummy dataset details for entropy calculation illustration

Let us assume a **dummy dataset** with

Number of samples – 10 (say s1, s2, s3,……s10)
Number of classes – 2 (c1, c2)
Number of attributes – not needed for calculating entropy (can be any number)

Assume k = 3 for the above dataset (best chosen 'k' after running k-means)

Dataset after running k-means may look like:

Sample Number	Assigned Label
S1	L1
S2	L2
S3	L1
S4	L2
S5	L3
S6	L2
S7	L3
S8	L2
S9	L2
S10	L3

As shown in the Table 10, dataset is assumed to have 10 samples or records. Each sample may have 'n' attributes where 'n' is unknown (not required for this illustration). Originally dataset had 2 labels (or classes) but after running k-means for wide range of 'k', it is determined that k=3 is good for the given dataset (this is assumed as it is dummy dataset). Dataset table in Table 10 is how the samples are assigned to three clusters after running k-means algorithm. Figure 11 shows the clustered samples.

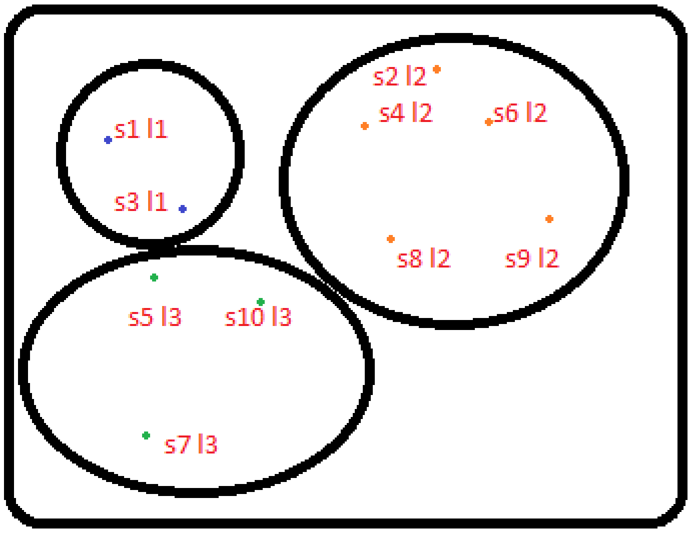


Figure 11: Clustering for dummy dataset

It is clearly seen from Figure 11 that there are 3 clusters with each cluster containing homogenous samples. Now Figure 12 shows the clustering result after performing the reduction (using any reduction technique).

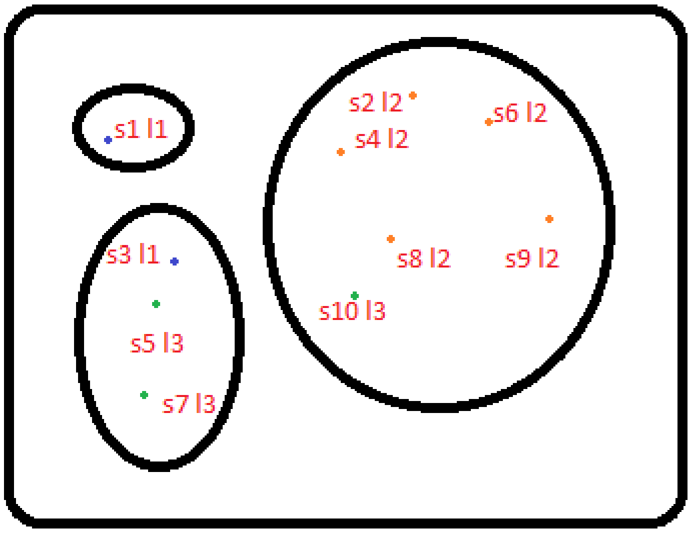


Figure 12: Clustering for dummy dataset after reduction

It is again clear from Figure 12 that there are 3 clusters but this time clusters do not contain homogenous samples. It is bit different from the original result. Hence we use weighted cluster entropy to check how the data structure is preserved.

$$\boxed{Entropy(S) = -\sum_{j=1}^{C} p(S,j) \times \log p(S,j)}$$

Equation 1: Individual Cluster Entropy Formula

In Equation 1,

P(S, j) refers to proportion of instances in cluster ‘S’ that belong to class label ‘j’

C refers to number of class labels

Entropy in Equation 1 is the individual entropy for each cluster. Aggregate entropy or final entropy for the data after reduction is calculated using the formula given in Equation 2.

$$\boxed{Aggregate\ Entropy = \sum_{j=1}^{L} \frac{|C_j|}{|C|} + Entropy\,(C_j)}$$

Equation 2: Aggregate or Final Entropy Formula

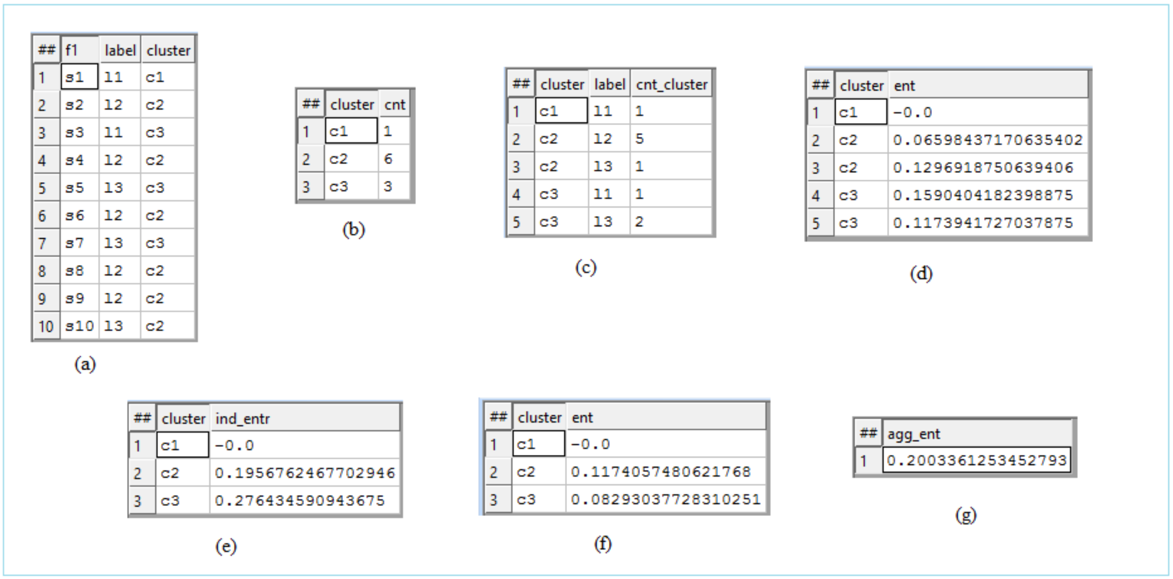
In Equation 2,

|C| refers to number of samples in all the clusters

|Cj| refers to number of samples in cluster Cj

L refers to number of clusters

Entropy(Cj) refers to individual entropy calculated using Equation 1 for each cluster Cj



##	f1	label	cluster
1	s1	l1	c1
2	s2	l2	c2
3	s3	l1	c3
4	s4	l2	c2
5	s5	l3	c3
6	s6	l2	c2
7	s7	l3	c3
8	s8	l2	c2
9	s9	l2	c2
10	s10	l3	c2

(a)

##	cluster	cnt
1	c1	1
2	c2	6
3	c3	3

(b)

##	cluster	label	cnt_cluster
1	c1	l1	1
2	c2	l2	5
3	c2	l3	1
4	c3	l1	1
5	c3	l3	2

(c)

##	cluster	ent
1	c1	-0.0
2	c2	0.06598437170635402
3	c2	0.1296918750639406
4	c3	0.1590404182398875
5	c3	0.1173941727037875

(d)

##	cluster	ind_entr
1	c1	-0.0
2	c2	0.1956762467702946
3	c3	0.276434590943675

(e)

##	cluster	ent
1	c1	-0.0
2	c2	0.1174057480621768
3	c3	0.08293037728310251

(f)

##	agg_ent
1	0.2003361253452793

(g)

Figure 13: Entropy Calculation Results in ECL

Figure 13 shows the results obtained for calculating entropy using ECL.

Figure 13(a) is the input dataset with 10 samples. There are 3 labels (l1, l2, l3) and 3 clusters (c1, c2, c3).

Figure 13(b) is the count of number of samples in each of the 3 clusters.

Figure 13(c) is the count of samples with some label 'l' that belong to some cluster 'c'. There is one sample labeled 'l1' in cluster 'c1', five samples labeled 'l2' in cluster 'c2', one sample labeled 'l3' in cluster 'c2' etc…

Figure 13(d) is the intermediate result in calculating individual entropy for each cluster.

Figure 13(e) shows individual entropy for each cluster.

Figure 13(f) is the intermediate result in calculating aggregate entropy.

Figure 13(g) is the final entropy or aggregate entropy.

Value of entropy always measure between 0 to 1. If entropy is low, it means that the amount of information loss is less and structure of the data is preserved. But if the entropy is high, it means that the data structure is not preserved after dimension reduction. When all the clusters contain homogenous data, entropy is '0'.

3.4 Proposed Approach Vs Existing Approach

From related work in Table 4, it is clear that many of the research works use reconstruction as a step to evaluate the structure of data after dimension reduction. After reconstructing or approximating the reduced data to original dimensions, error is computed which can be reconstruction error or root mean squared error or reconstruction accuracy. But as stated in [12], reconstruction error may not always suitable for checking the local structure of the data. Higher reconstruction error may not result in poor dimension reduction always.

Every technique has its own algorithm to reconstruct the reduced data to original data. For instance, if PCA is considered, say 'Z' is the reduced data, then approximated original data say 'X' is given by the matrix product of U_{reduce} and Z. The time complexity for calculating the approximate matrix or reconstructed matrix is same as the original algorithm. If stacked auto-encoder is taken as another instance, during reduction process, the network learns its representation in an encoded form by compressing the data and extracting only the important features required. But during expansion process or reconstruction, decoding is performed which is again repeating the whole algorithm. In real-time, network architecture is usually deep which means encoding and decoding process takes a lot of time and practically infeasible. Hence this process of encoder-decoder is performed for auto-encoders because they may not have deep architectures but for stacked auto-encoders, it is really a complex process to repeat the same and even if we refer to the evaluation in [25], it is just visual comparison. Similarly reconstruction process is a tedious process in other approaches also like SVD and Deep Belief Networks. But the evaluation cannot be performed by just reconstruction, squared error difference is calculated between the original data and the reconstructed data which is treated as a measure for comparison. So, this kind of evaluation involves many steps and may take longer time in case of real-world big datasets.

In the proposed approach, we use k-means algorithm to check the clustering before and after reduction and use the cluster labels for calculating the entropy. Running k-means is scalable and easy even for big data. In real-world datasets, we do not have original labels which means unsupervised methods are essential for analysing such data sets. K-means is one of the best unsupervised approaches which helps in finding the similar data and it is reasonably fast except in its worst case. Key part in k-means algorithm is choosing ‘k’ which plays a significant role. A method to choose ‘k’ is also given in the proposed approach.

Thus, we choose unsupervised clustering based approach to evaluate structure of the data after reduction rather than complex process of reconstruction.

3.5 Dimension Reduction Approaches for Comparative Study With Respect To Visualization

SVD (Singular Value Decomposition):

As mentioned in chapter 2, in order to reduce the dimensions for the original data, we need to take the subset of rows and columns of U, Σ and V to get the lower order matrices.

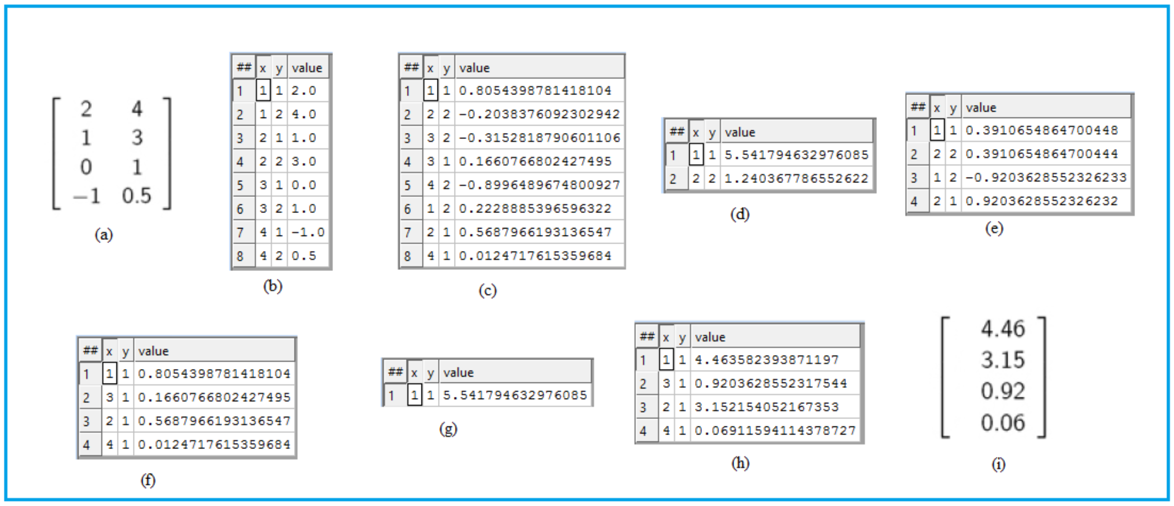
If we need to reduce the data from ‘n’ dimensions to 3 dimensions, then

U_{prime} = Use only initial 3 columns for U and

Σ_{prime} = Use only 3 values for Σ

Now the lower order matrix or 3-dimensional data for the original data is given by $U_{prime} * \Sigma_{prime}$

Figure 14 takes an example of some random data with two dimensions and it is reduced to one dimension using SVD.



$$\begin{bmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 1 \\ -1 & 0.5 \end{bmatrix}$$

(a)

##	x	y	value
1	1	1	2.0
2	1	2	4.0
3	2	1	1.0
4	2	2	3.0
5	3	1	0.0
6	3	2	1.0
7	4	1	-1.0
8	4	2	0.5

(b)

##	x	y	value
1	1	1	0.8054398781418104
2	2	2	-0.2038376092302942
3	3	2	-0.3152818790601106
4	3	1	0.1660766802427495
5	4	2	-0.8996489674800927
6	1	2	0.2228885396596322
7	2	1	0.5687966193136547
8	4	1	0.0124717615359684

(c)

##	x	y	value
1	1	1	5.541794632976085
2	2	2	1.240367786552622

(d)

##	x	y	value
1	1	1	0.3910654864700448
2	2	2	0.3910654864700444
3	1	2	-0.9203628552326233
4	2	1	0.9203628552326232

(e)

##	x	y	value
1	1	1	0.8054398781418104
2	3	1	0.1660766802427495
3	2	1	0.5687966193136547
4	4	1	0.0124717615359684

(f)

##	x	y	value
1	1	1	5.541794632976085

(g)

##	x	y	value
1	1	1	4.463582393871197
2	3	1	0.9203628552317544
3	2	1	3.152154052167353
4	4	1	0.06911594114378727

(h)

$$\begin{bmatrix} 4.46 \\ 3.15 \\ 0.92 \\ 0.06 \end{bmatrix}$$

(i)

Figure 14: ECL Results for SVD for random dataset

Figure 14 (a) - matrix representation for input data with 2 dimensions

Figure 14 (b) - input dataset with two dimensions in ECL output form

Figure 14 (c) - unitary matrix U in singular value decomposition factorization

Figure 14 (d) - diagonal matrix Σ in singular value decomposition factorization

Figure 14 (e) - unitary matrix V in singular value decomposition factorization

Figure 14 (f) - unitary matrix U_{prime}, lowered from U considering only first column

Figure 14 (g) - diagonal matrix Σ_{prime}, lowered from Σ considering only one value

Figure 14 (h) - Product of U_{prime} and Σ_{prime} that gives the reduced data with one dimension

Figure 14 (i) - Matrix representation for Figure 14 (h)

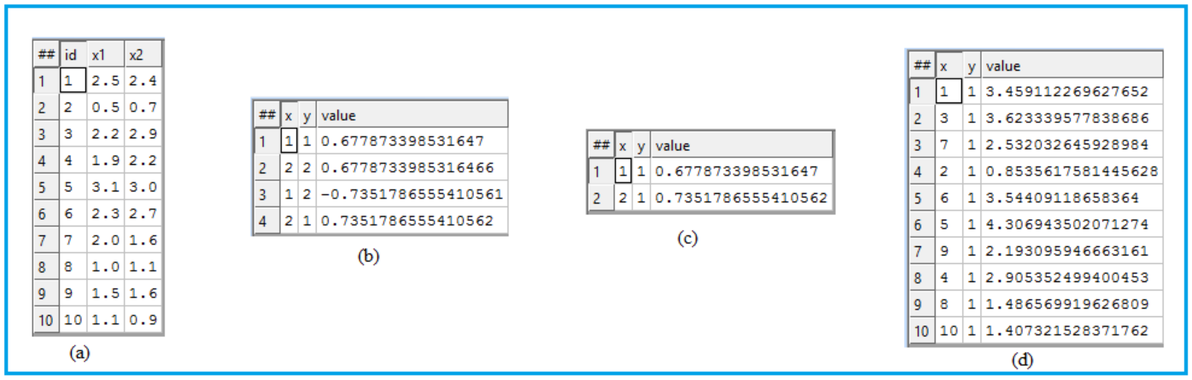
PCA (Principal Component Analysis):

Again little introduction to PCA is already given in chapter 2. But here, description is given on how to reduce the data to 3 dimensions.

We need to consider initial 3 principal components from the 'n' principal components that are obtained after running PCA for the original dataset. The reduced principal components or subset of principal components is represented as "U_{reduce}". Now the reduced dataset is given by the product of original matrix with U_{reduce} which is X * U_{reduce}.

Here X has dimensions as number_of_samples * number_of_dimensions and product of X and U_{reduce} is 3-dimensional in our scenario.

Figure 15 takes an example of some random data with two dimensions and it is reduced to one dimension using PCA.



##	id	x1	x2
1	1	2.5	2.4
2	2	0.5	0.7
3	3	2.2	2.9
4	4	1.9	2.2
5	5	3.1	3.0
6	6	2.3	2.7
7	7	2.0	1.6
8	8	1.0	1.1
9	9	1.5	1.6
10	10	1.1	0.9

(a)

##	x	y	value
1	1	1	0.677873398531647
2	2	2	0.6778733985316466
3	1	2	-0.7351786555410561
4	2	1	0.7351786555410562

(b)

##	x	y	value
1	1	1	0.677873398531647
2	2	1	0.7351786555410562

(c)

##	x	y	value
1	1	1	3.459112269627652
2	3	1	3.623339577838686
3	7	1	2.532032645928984
4	2	1	0.8535617581445628
5	6	1	3.54409118658364
6	5	1	4.306943502071274
7	9	1	2.193095946663161
8	4	1	2.905352499400453
9	8	1	1.486569919626809
10	10	1	1.407321528371762

(d)

Figure 15: ECL results for PCA for random dataset

Figure 15 (a) - input dataset with two dimensions in ECL output form

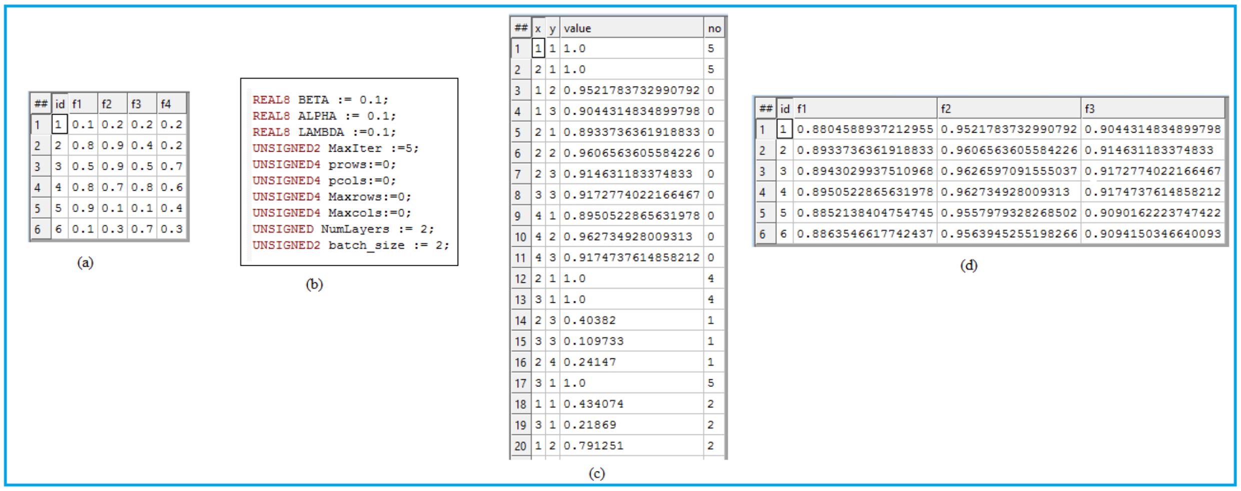
Figure 15 (b) - two principal components for the two dimensional dataset

Figure 15 (c) - one principal component which is U_{reduce} (reduced form of Figure 15 (b))

Figure 15 (d) - Two dimensions in the original dataset reduced to one dimension

DBN (Deep Belief Networks):

DBN is spoken technically in terms of its algorithm in chapter 5. Using the same algorithmic approach, this section explains how dimension reduction is performed.



##	id	f1	f2	f3	f4
1	1	0.1	0.2	0.2	0.2
2	2	0.8	0.9	0.4	0.2
3	3	0.5	0.9	0.5	0.7
4	4	0.8	0.7	0.8	0.6
5	5	0.9	0.1	0.1	0.4
6	6	0.1	0.3	0.7	0.3

(a)

```
REAL8 BETA := 0.1;
REAL8 ALPHA := 0.1;
REAL8 LAMBDA :=0.1;
UNSIGNED2 MaxIter :=5;
UNSIGNED4 prows:=0;
UNSIGNED4 pcols:=0;
UNSIGNED4 Maxrows:=0;
UNSIGNED4 Maxcols:=0;
UNSIGNED NumLayers := 2;
UNSIGNED2 batch_size := 2;
```

(b)

##	x	y	value	no
1	1	1	1.0	5
2	2	1	1.0	5
3	1	2	0.9521783732990792	0
4	1	3	0.9044314834899798	0
5	2	1	0.8933736361918833	0
6	2	2	0.9606563605584226	0
7	2	3	0.914631183374833	0
8	3	3	0.9172774022166467	0
9	4	1	0.8950522865631978	0
10	4	2	0.962734928009313	0
11	4	3	0.9174737614858212	0
12	2	1	1.0	4
13	3	1	1.0	4
14	2	3	0.40382	1
15	3	3	0.109733	1
16	2	4	0.24147	1
17	3	1	1.0	5
18	1	1	0.434074	2
19	3	1	0.21869	2
20	1	2	0.791251	2

(c)

##	id	f1	f2	f3
1	1	0.8804588937212955	0.9521783732990792	0.9044314834899798
2	2	0.8933736361918833	0.9606563605584226	0.914631183374833
3	3	0.8943029937510968	0.9626597091555037	0.9172774022166467
4	4	0.8950522865631978	0.962734928009313	0.9174737614858212
5	5	0.8852138404754745	0.9557979328268502	0.9090162223747422
6	6	0.8863546617742437	0.9563945255198266	0.9094150346640093

(d)

Figure 16: ECL results for DBN for random dataset

Figure 16 (a) input dataset with four dimensions in ECL output form

Figure 16 (b) parameters used and their values for executing DBN

Figure 16 (c) Learnt Model for the given input with weights and bias (complete model is not shown in the figure, only part of it is presented)

Figure 16 (d) Final output with reduced dimensions to three from original four

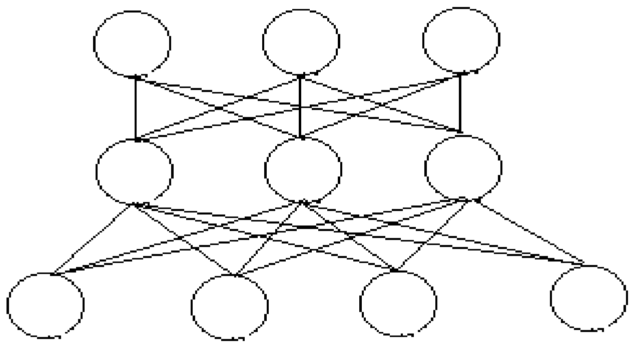
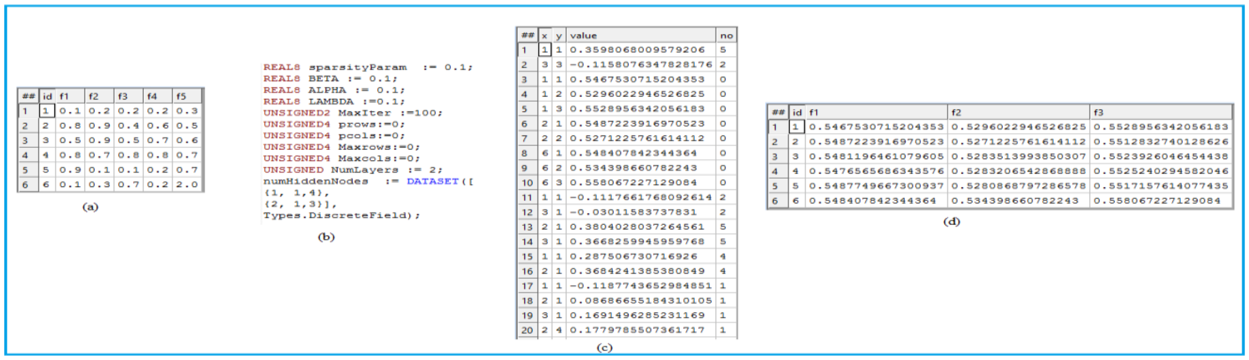


Figure 17: Network architecture for DBN

Now using Figure 16 and Figure 17, dimension reduction using DBN can be explained. The dataset used is just for illustration purposes. Hence it has less number of samples and dimensions. Thus, the architecture is shallow like 4 nodes in the input layer, 3 nodes in hidden layer and 3 nodes in output layer. For the real-time datasets it may be deeper than the architecture specified in Figure 17. The number of dimensions to which the original dataset should be reduced depends on the architecture specified. Here the architecture is 4-3-3. Thus the output from DBN is 3-dimensional whereas the input fed to it is 4-dimensional. The learning technique called contrastive divergence used in DBN, getting the final output and every other detail about how algorithm works is mentioned in chapter 5. Since the batch_size is mentioned as 2 (from Figure 16), input data is processed as 3 batches with each batch having 2 samples.

Stacked Auto-encoders:

Stacked Auto-encoders work very similar to DBN in terms of stacking but the algorithm or learning technique is different. Auto-encoders are stacked together to form stacked auto-encoders. Again dimension reduction is same, the number of dimensions to which the original dataset should be reduced depends on the architecture specified.



##	id	f1	f2	f3	f4	f5
1	1	0.1	0.2	0.2	0.2	0.3
2	2	0.8	0.9	0.4	0.6	0.5
3	3	0.5	0.9	0.5	0.7	0.6
4	4	0.8	0.7	0.8	0.8	0.7
5	5	0.9	0.1	0.1	0.2	0.7
6	6	0.1	0.3	0.7	0.2	2.0

(a)

```
REAL8 sparsityParam  := 0.1;
REAL8 BETA := 0.1;
REAL8 ALPHA := 0.1;
REAL8 LAMBDA :=0.1;
UNSIGNED2 MaxIter :=100;
UNSIGNED4 prows:=0;
UNSIGNED4 pcols:=0;
UNSIGNED4 Maxrows:=0;
UNSIGNED4 Maxcols:=0;
UNSIGNED NumLayers := 2;
numHiddenNodes  := DATASET([
{1, 1,4},
{2, 1,3}],
Types.DiscreteField);
```

(b)

##	x	y	value	no
1	1	1	0.3598068009579206	5
2	3	3	-0.1158076347828176	2
3	1	1	0.5467530715204353	0
4	1	2	0.5296022946526825	0
5	1	3	0.5528956342056183	0
6	2	1	0.5487223916970523	0
7	2	2	0.5271225761614112	0
8	6	1	0.548407842344364	0
9	6	2	0.534398660782243	0
10	6	3	0.558067227129084	0
11	1	1	-0.1117661768092614	2
12	3	1	-0.03011583737831	2
13	2	1	0.3804028037264561	5
14	3	1	0.3668259945959768	5
15	1	1	0.287506730716926	4
16	2	1	0.3684241385380849	4
17	1	1	-0.1187743652984851	1
18	2	1	0.08686655184310105	1
19	3	1	0.1691496285231169	1
20	2	4	0.1779785507361717	1

(c)

##	id	f1	f2	f3
1	1	0.5467530715204353	0.5296022946526825	0.5528956342056183
2	2	0.5487223916970523	0.5271225761614112	0.5512832740128626
3	3	0.5481196461079605	0.5283513993850307	0.5523926046454438
4	4	0.5476565686343576	0.5283206542868888	0.5525240294582046
5	5	0.5487749667300937	0.5280868797286578	0.5517157614077435
6	6	0.548407842344364	0.534398660782243	0.558067227129084

(d)

Figure 18: ECL results for stacked auto-encoders for random dataset

Figure 18 (a) input dataset with five dimensions in ECL output form

Figure 18 (b) parameters used and their values for executing stacked auto-encoders

Figure 18 (c) Learnt Model for the given input with weights and bias (complete model not shown in the figure, only a part of it is presented)

Figure 18 (d) Final output with reduced dimensions to three from original five

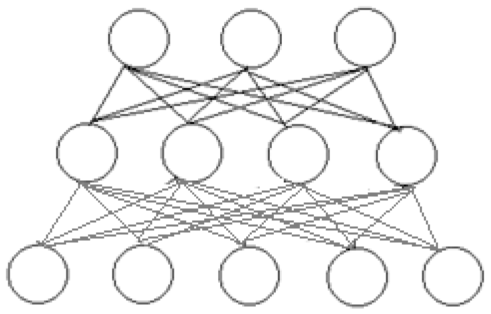


Figure 19: Network architecture for Stacked Auto-encoders

Network architecture chosen for running stacked auto-encoders is 5-4-3 for given random dataset. There is no particular rule to choose the network architecture or the number of nodes in each of the layers. For the real-time datasets, number of experiments are performed and the best parameters and network architecture is presented and same is used for comparison. Since the number of nodes in output layer is 3, the reduced data or output has 3-dimensions.

In the next chapter, real-time datasets are used for dimension reduction with all the four approaches (two traditional and two deep learning). Comparison results along with visualization are also presented in the following chapter.

CHAPTER 4

Experimental Study

4.1 Overview

In this Chapter, we demonstrate example datasets for dimension reduction with the proposed approach discussed in Chapter 3 and using HPCC deep learning module. We evaluate the results and produce a conclusion about which dimension reduction technique is suitable and gives best results for the datasets used. Moreover, we demonstrate our approach clearly with outputs obtained at each step.

Datasets used for illustration are Ionosphere [37], Breast Cancer Wisconsin (Original) [38, 39], Wine [40, 41], Shuttle Landing Control [42] and Ecoli [43]. For each dataset, results of all four dimension reduction techniques are produced and finally comparison is performed.

Table 11 shows the number of attributes, number of classes, number of instances, normalization details for all the datasets but in further sections details are provided more clearly.

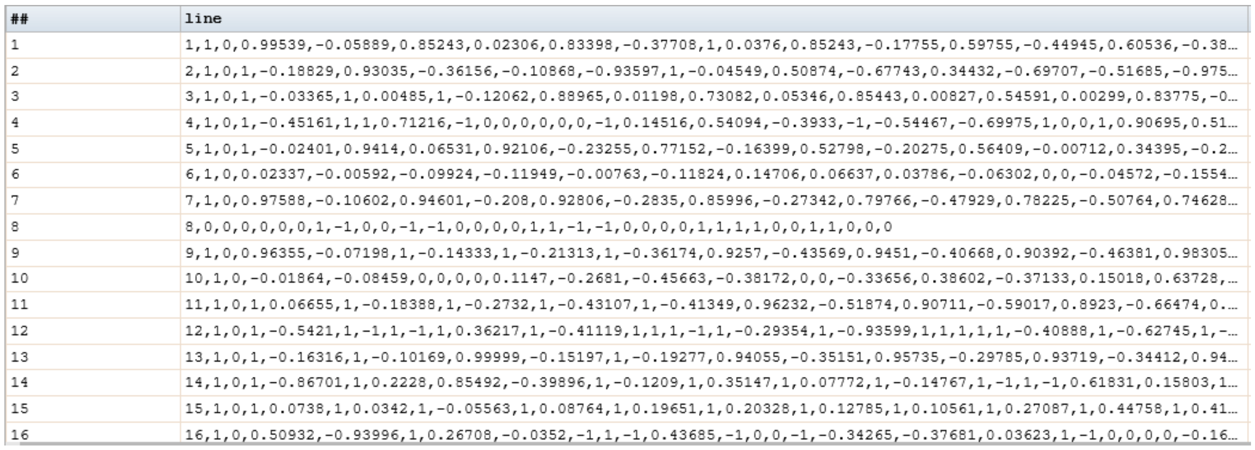
Table 11: Datasets and relevant information

Datasets	Number of Dimensions	Number of classes	Number of instances	Normalized or Original
Ionosphere [37]	34	2	351	Original
Breast Cancer Wisconsin (Original) [38, 39]	9	2	699	Normalized using min-max normalization form 2
Wine [40, 41]	13	3	178	Normalized using min-max normalization form 1
Shuttle Landing Control [42]	6	2	278	Normalized using min-max normalization form 1
Ecoli [43]	7	8	336	Original

Table 11 gives an idea that datasets with different characteristics are considered and chosen for experimental study.

4.2 Dataset 1: Ionosphere

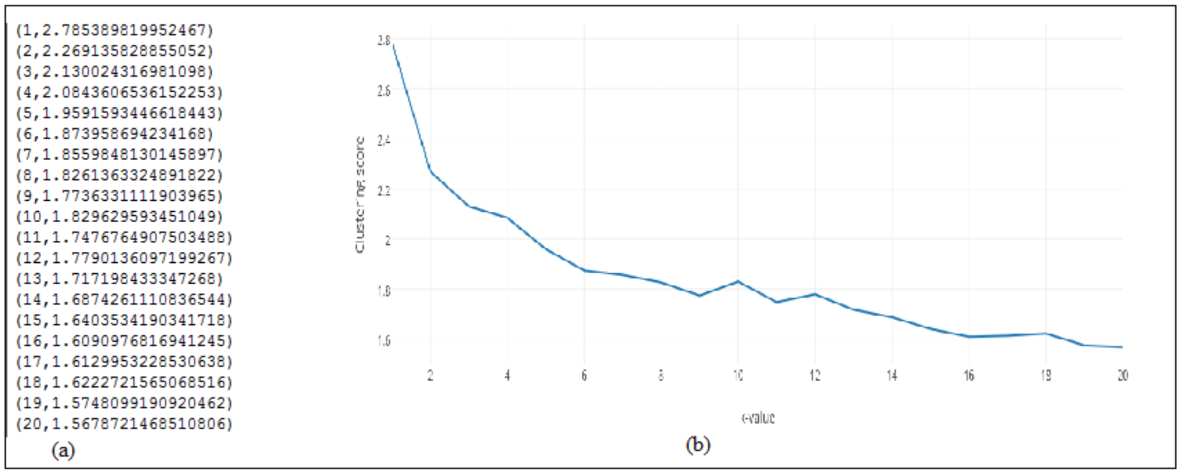
This contains radar data collected by a system in Goose Bay, Labrador [44]. Snapshot of data is presented in Figure 20. Since the data is within a specific range [-1, 1], original data is used for reduction. Normalization is not required for such data since there are no extreme values that may have more weight over the other while running the algorithm.



##	line
1	1,1,0,0.99539,-0.05889,0.85243,0.02306,0.83398,-0.37708,1,0.0376,0.85243,-0.17755,0.59755,-0.44945,0.60536,-0.38…
2	2,1,0,1,-0.18829,0.93035,-0.36156,-0.10868,-0.93597,1,-0.04549,0.50874,-0.67743,0.34432,-0.69707,-0.51685,-0.975…
3	3,1,0,1,-0.03365,1,0.00485,1,-0.12062,0.88965,0.01198,0.73082,0.05346,0.85443,0.00827,0.54591,0.00299,0.83775,-0…
4	4,1,0,1,-0.45161,1,1,0.71216,-1,0,0,0,0,0,-1,0.14516,0.54094,-0.3933,-1,-0.54467,-0.69975,1,0,0,1,0.90695,0.51…
5	5,1,0,1,-0.02401,0.9414,0.06531,0.92106,-0.23255,0.77152,-0.16399,0.52798,-0.20275,0.56409,-0.00712,0.34395,-0.2…
6	6,1,0,0.02337,-0.00592,-0.09924,-0.11949,-0.00763,-0.11824,0.14706,0.06637,0.03786,-0.06302,0,0,-0.04572,-0.1554…
7	7,1,0,0.97588,-0.10602,0.94601,-0.208,0.92806,-0.2835,0.85996,-0.27342,0.79766,-0.47929,0.78225,-0.50764,0.74628…
8	8,0,0,0,0,0,0,1,-1,0,0,-1,-1,0,0,0,0,1,1,-1,-1,0,0,0,0,1,1,1,1,0,0,1,1,0,0,0
9	9,1,0,0.96355,-0.07198,1,-0.14333,1,-0.21313,1,-0.36174,0.9257,-0.43569,0.9451,-0.40668,0.90392,-0.46381,0.98305…
10	10,1,0,-0.01864,-0.08459,0,0,0,0,0.1147,-0.2681,-0.45663,-0.38172,0,0,-0.33656,0.38602,-0.37133,0.15018,0.63728,…
11	11,1,0,1,0.06655,1,-0.18388,1,-0.2732,1,-0.43107,1,-0.41349,0.96232,-0.51874,0.90711,-0.59017,0.8923,-0.66474,0.…
12	12,1,0,1,-0.5421,1,-1,1,-1,1,0.36217,1,-0.41119,1,1,1,-1,1,-0.29354,1,-0.93599,1,1,1,1,1,-0.40888,1,-0.62745,1,-…
13	13,1,0,1,-0.16316,1,-0.10169,0.99999,-0.15197,1,-0.19277,0.94055,-0.35151,0.95735,-0.29785,0.93719,-0.34412,0.94…
14	14,1,0,1,-0.86701,1,0.2228,0.85492,-0.39896,1,-0.1209,1,0.35147,1,0.07772,1,-0.14767,1,-1,1,-1,0.61831,0.15803,1…
15	15,1,0,1,0.0738,1,0.0342,1,-0.05563,1,0.08764,1,0.19651,1,0.20328,1,0.12785,1,0.10561,1,0.27087,1,0.44758,1,0.41…
16	16,1,0,0.50932,-0.93996,1,0.26708,-0.0352,-1,1,-1,0.43685,-1,0,0,-1,-0.34265,-0.37681,0.03623,1,-1,0,0,0,0,-0.16…

Figure 20: Snapshot of Ionosphere Dataset

Since normalization is not required, our next step is to find the good value for 'k' by running k-means over a wide range of 'k' say 1 to 20, as original classes are just 2, this range would be sufficient.



```
(1,2.785389819952467)
(2,2.269135828855052)
(3,2.130024316981098)
(4,2.0843606536152253)
(5,1.9591593446618443)
(6,1.873958694234168)
(7,1.8559848130145897)
(8,1.8261363324891822)
(9,1.7736331111903965)
(10,1.829629593451049)
(11,1.7476764907503488)
(12,1.7790136097199267)
(13,1.717198433347268)
(14,1.6874261110836544)
(15,1.6403534190341718)
(16,1.6090976816941245)
(17,1.6129953228530638)
(18,1.6222721565068516)
(19,1.5748099190920462)
(20,1.5678721468510806)
```

Figure 21: Clustering Score and its graph for Ionosphere dataset

Figure 21 (a) refers to clustering score for k-values from 1 to 20

Figure 21 (b) graph plotted for k-values and their clustering scores in Figure 21 (a)

From graph in Figure 21(b), it is observed that there is a sharp change in clustering score at k-value 2 and later the change is not clearly noticeable. Hence best 'k' is chosen to be '2' for ionosphere dataset and clustering is performed.

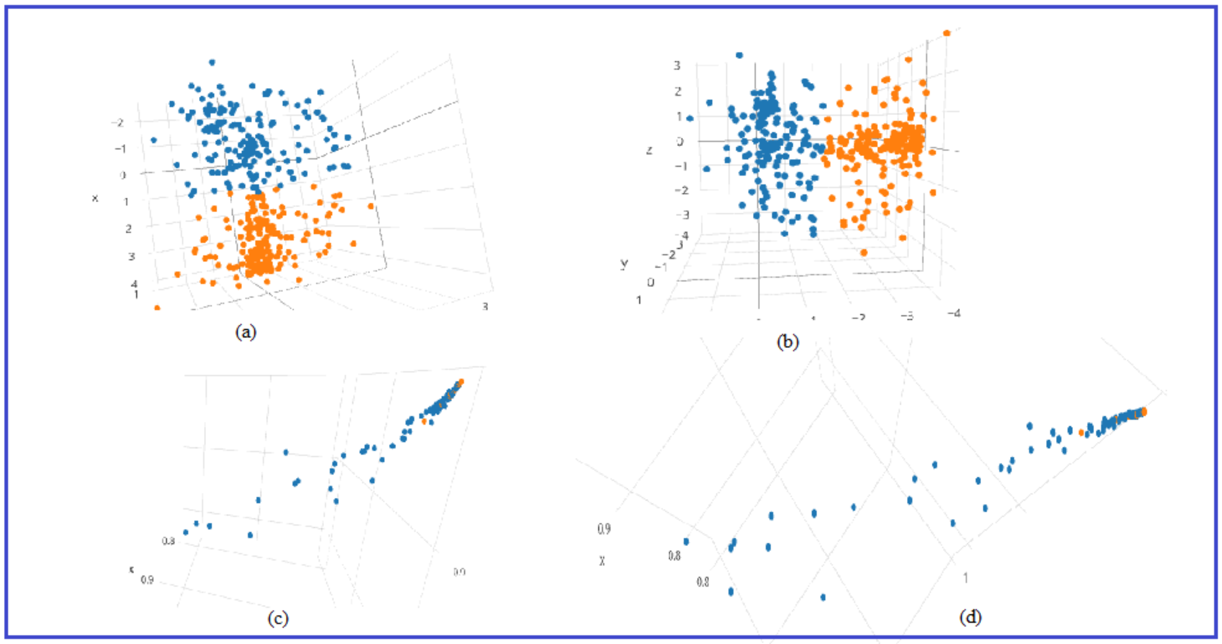


Figure 22: Visualization for ionosphere reduced 3-dimensional dataset

Figure 22 (a) - visualization for PCA reduced dataset

Figure 22 (b) - visualization for SVD reduced dataset

Figure 22 (c) - visualization for DBN reduced dataset

Figure 22 (d) - visualization for stacked auto-encoders reduced dataset

From the visualization observations using Figure 22, there is overlapping in clustering using every technique except PCA and SVD. The visualization for PCA and SVD show two separable classes, hence the loss of information should be less in such a case when compared to others. This can be verified by entropy results in Table 12.

Table 12 shows the aggregate entropy values for all the four dimension reduction techniques.

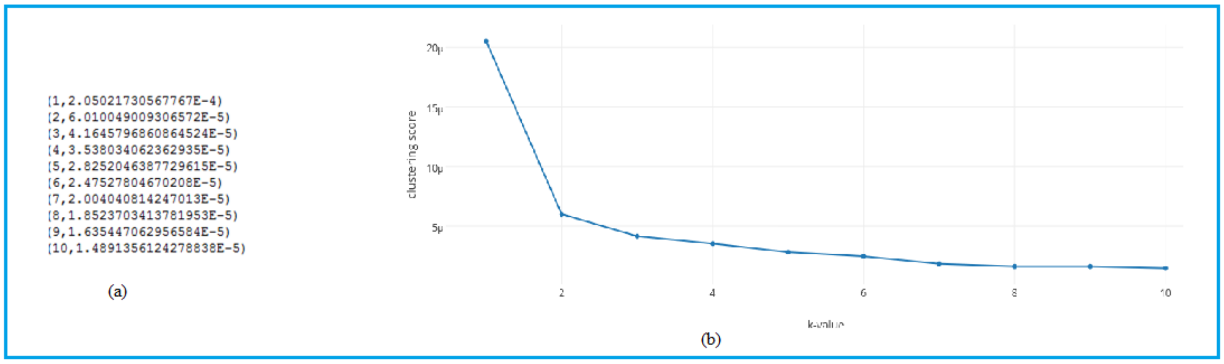
Table 12: Entropy values for Ionosphere dataset

Reduction Technique	**Entropy Value**
PCA	0.01524692879907179
SVD	0.02091038972074466
Deep Belief Networks	0.2916949005086614
Stacked Auto Encoders	0.2906937855343454

From the results in Table 12, entropy for PCA is low and then comes SVD with little difference, hence its visualization is clear and we can see that two classes are separable. Our statement that PCA and SVD may perform better just by seeing visualization is confirmed through entropy determination.

4.3 Dataset 2: Breast Cancer Wisconsin (Original)

Details about breast cancer dataset is already provided in chapter 1 (Table 1 and Figure 3). Visualization and entropy results are provided in this section, but before that best k-value has to be chosen. Figure 23 helps in choosing k-value for this dataset.



```
(1,2.05021730567767E-4)
(2,6.010049009306572E-5)
(3,4.1645796860864524E-5)
(4,3.538034062362935E-5)
(5,2.8252046387729615E-5)
(6,2.47527804670208E-5)
(7,2.004040814247013E-5)
(8,1.8523703413781953E-5)
(9,1.635447062956584E-5)
(10,1.489135612427883E-5)
```

Figure 23: Clustering Score and its graph for Breast Cancer dataset

Figure 23 (a) - shows clustering scores for k-values starting from 1 to 10.

Figure 23 (b) - shows graph for Figure 23 (a)

From the graph in Figure 23 (b), it is again a sharp decrease seen at k-value 2 but later decrease is not significantly noticeable. Hence best chosen k-value for this dataset is 2.

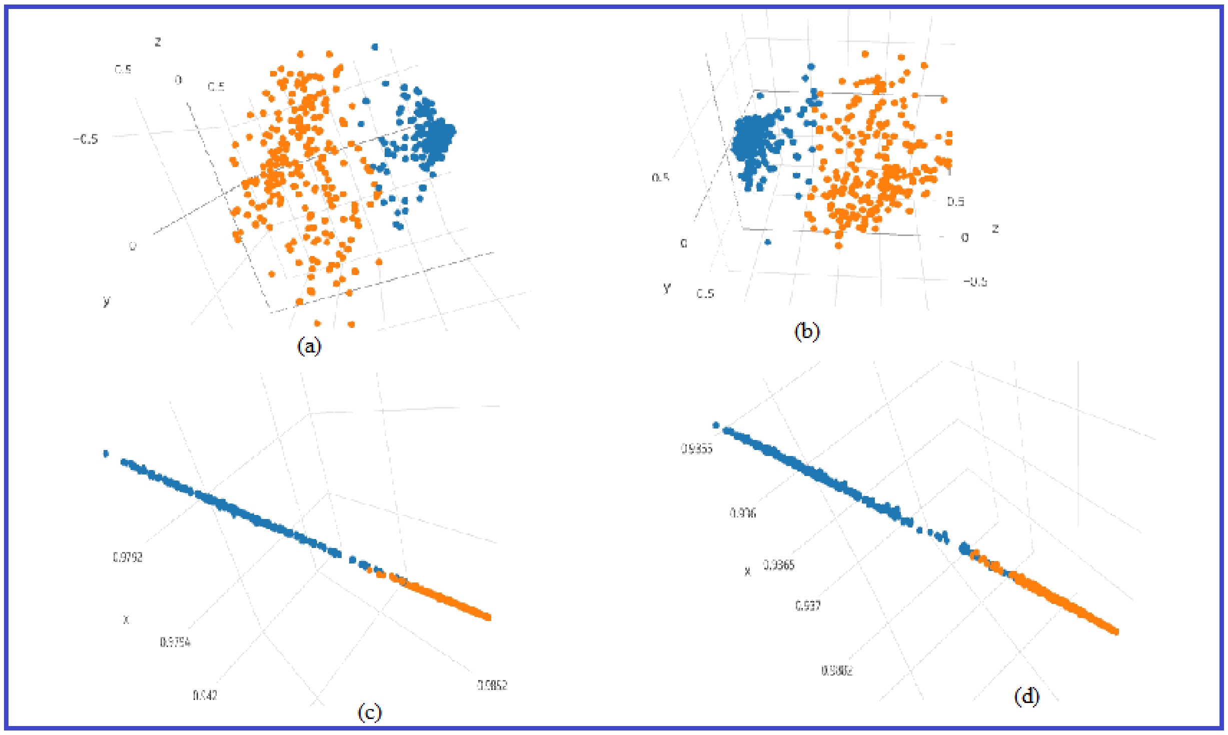


Figure 24: Visualizations for reduced breast cancer 3-dimensional dataset

Figure 24 (a) - visualization for PCA reduced dataset
Figure 24 (b) - visualization for SVD reduced dataset
Figure 24 (c) - visualization for DBN reduced dataset
Figure 24 (d) - visualization for stacked auto-encoders reduced dataset

All the visualization representations have separable classes in Figure 24, it is hard to distinguish which technique performed better in this case. Hence to determine the entropy is critical step which decides the best technique suitable for this dataset.
Table 13 gives the aggregate entropy values for all the four datasets.

Table 13: Entropy values for breast cancer dataset

Reduction Technique	**Entropy Value**
PCA	0.0

SVD	0.004439407792323756
DBN	0.05664015277201884
Stacked Auto-encoders	0.06216426790608801

From the results in Table 13, though it can be said that PCA performed better since its entropy is 0, even other techniques have low entropy. Hence loss of information is less even after reduction for this dataset.

4.4 Dataset 3: Wine

The data is from the results of chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines [44].

Table 14 shows the attribute information for wine dataset.

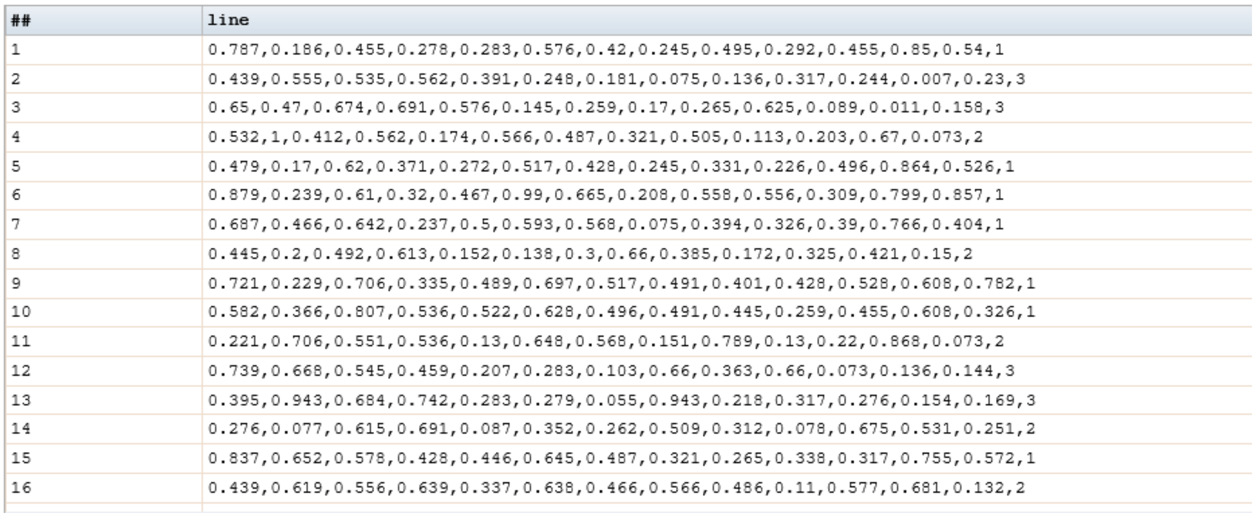
Table 14: Attribute information for wine dataset

Number of Attributes : 13
Number of classes : 3
Number of Samples : 178

Attribute Information:

1. Alcohol
2. Malic Acid
3. Ash
4. Alcalinity of ash
5. Magnesium
6. Total phenols
7. Flavanoids
8. Nonflavanoid phenols
9. Proanthocyanins
10. Color intensity
11. Hue
12. OD280/OD315 of diluted wines
13. Proline

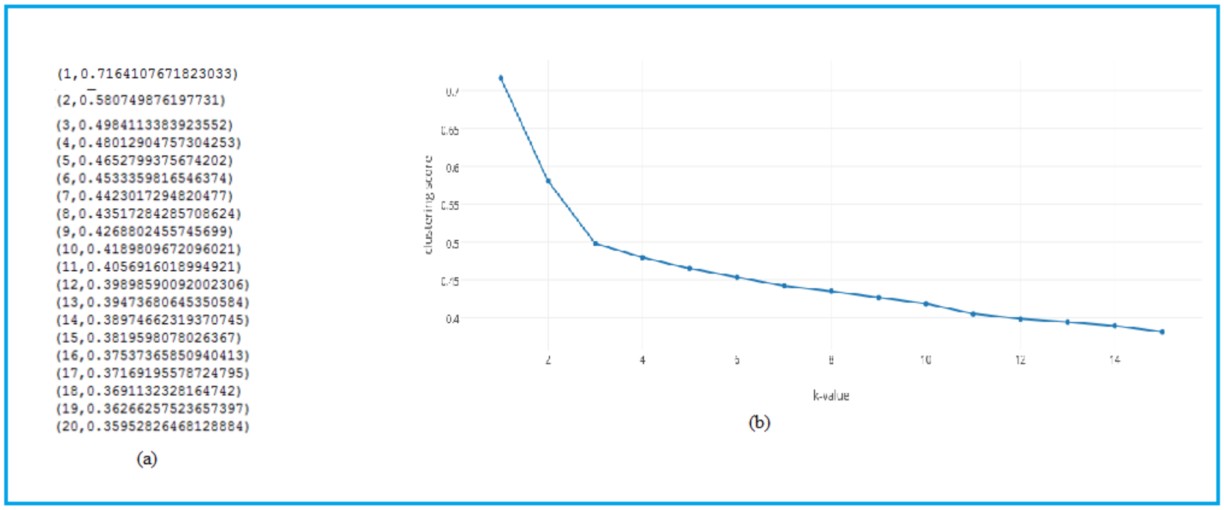
1, 2, 3 (3 types of wines) – Classes



##	line
1	0.787,0.186,0.455,0.278,0.283,0.576,0.42,0.245,0.495,0.292,0.455,0.85,0.54,1
2	0.439,0.555,0.535,0.562,0.391,0.248,0.181,0.075,0.136,0.317,0.244,0.007,0.23,3
3	0.65,0.47,0.674,0.691,0.576,0.145,0.259,0.17,0.265,0.625,0.089,0.011,0.158,3
4	0.532,1,0.412,0.562,0.174,0.566,0.487,0.321,0.505,0.113,0.203,0.67,0.073,2
5	0.479,0.17,0.62,0.371,0.272,0.517,0.428,0.245,0.331,0.226,0.496,0.864,0.526,1
6	0.879,0.239,0.61,0.32,0.467,0.99,0.665,0.208,0.558,0.556,0.309,0.799,0.857,1
7	0.687,0.466,0.642,0.237,0.5,0.593,0.568,0.075,0.394,0.326,0.39,0.766,0.404,1
8	0.445,0.2,0.492,0.613,0.152,0.138,0.3,0.66,0.385,0.172,0.325,0.421,0.15,2
9	0.721,0.229,0.706,0.335,0.489,0.697,0.517,0.491,0.401,0.428,0.528,0.608,0.782,1
10	0.582,0.366,0.807,0.536,0.522,0.628,0.496,0.491,0.445,0.259,0.455,0.608,0.326,1
11	0.221,0.706,0.551,0.536,0.13,0.648,0.568,0.151,0.789,0.13,0.22,0.868,0.073,2
12	0.739,0.668,0.545,0.459,0.207,0.283,0.103,0.66,0.363,0.66,0.073,0.136,0.144,3
13	0.395,0.943,0.684,0.742,0.283,0.279,0.055,0.943,0.218,0.317,0.276,0.154,0.169,3
14	0.276,0.077,0.615,0.691,0.087,0.352,0.262,0.509,0.312,0.078,0.675,0.531,0.251,2
15	0.837,0.652,0.578,0.428,0.446,0.645,0.487,0.321,0.265,0.338,0.317,0.755,0.572,1
16	0.439,0.619,0.556,0.639,0.337,0.638,0.466,0.566,0.486,0.11,0.577,0.681,0.132,2

Figure 25: Snapshot of normalized wine dataset

Figure 25 gives a snapshot of samples in wine dataset after normalization.



```
(1,0.7164107671823033)
(2,0.580749876197731)
(3,0.4984113383923552)
(4,0.48012904757304253)
(5,0.4652799375674202)
(6,0.4533359816546374)
(7,0.4423017294820477)
(8,0.43517284285708624)
(9,0.4268802455745699)
(10,0.4189809672096021)
(11,0.4056916018994921)
(12,0.39898590092002306)
(13,0.39473680645350584)
(14,0.38974662319370745)
(15,0.3819598078026367)
(16,0.37537365850940413)
(17,0.37169195578724795)
(18,0.3691132328164742)
(19,0.36266257523657397)
(20,0.35952826468128884)
```

Figure 26: Clustering Scores and its graph for wine dataset

Figure 26 (a) refers to clustering score for k-values from 1 to 20

Figure 26 (b) graph plotted for k-values and their clustering scores in Figure 26 (a)

From Figure 26, best k-value is chosen to be 3 because there is a noticeable decrease at two points (at 2 and at 3). After 3, decrease is not significant.

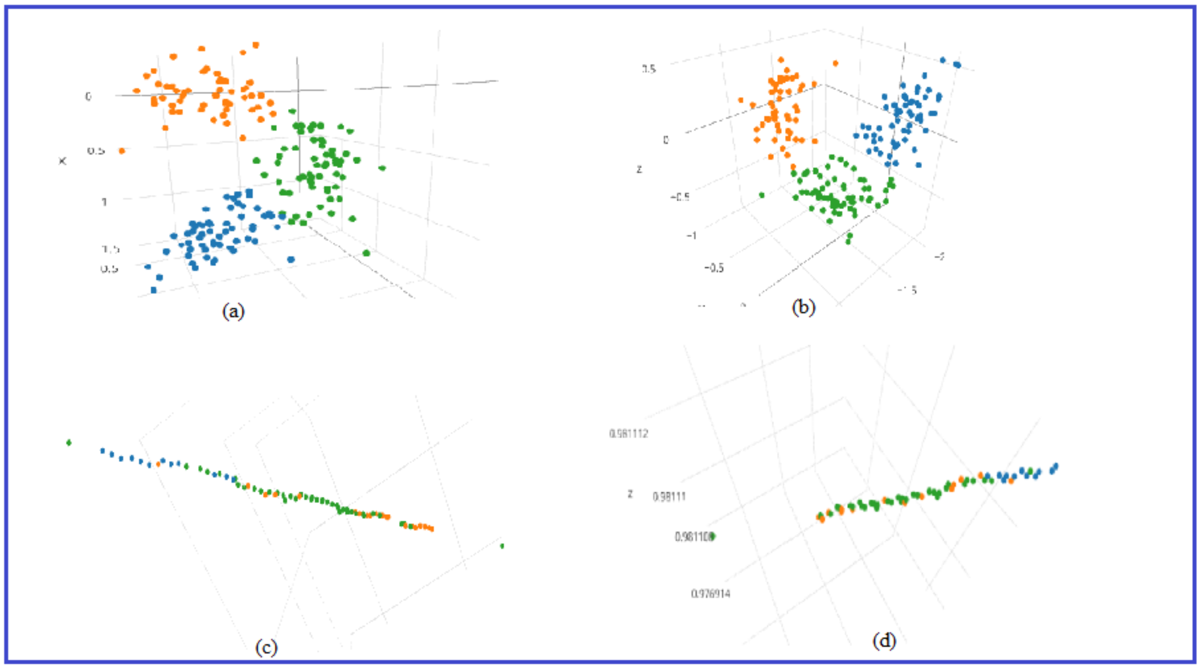


Figure 27: Visualizations for reduced wine 3-dimensional data

Figure 27 (a) - visualization for PCA reduced dataset

Figure 27 (b) - visualization for SVD reduced dataset

Figure 27 (c) - visualization for DBN reduced dataset

Figure 27 (d) - visualization for stacked auto-encoders reduced dataset

Figure 27 gives an idea of structure of reduced data through visualization. There is some overlapping found in Figure 27 (c) and Figure 27 (d) whereas other two are clearly separable. Same has to be validated through aggregate entropy values using Table 15.

Table 15: Entropy values for wine dataset

Reduction Technique	**Entropy Value**
PCA	0.06597000325091695
SVD	0.0
Deep Belief Networks	0.3394460069425066
Stacked Auto Encoders	0.3103455697842063

Thus visualization results are in sync with entropy results even for wine dataset. Entropy is low for SVD and PCA when compared to other two techniques.

4.5 Dataset 4: Shuttle Landing Control

This dataset has rules for generating comprehendible results for determining the conditions under which auto landing would be preferable to manual control of the space craft [44].

Attribute information for this dataset is provided in Table 16.

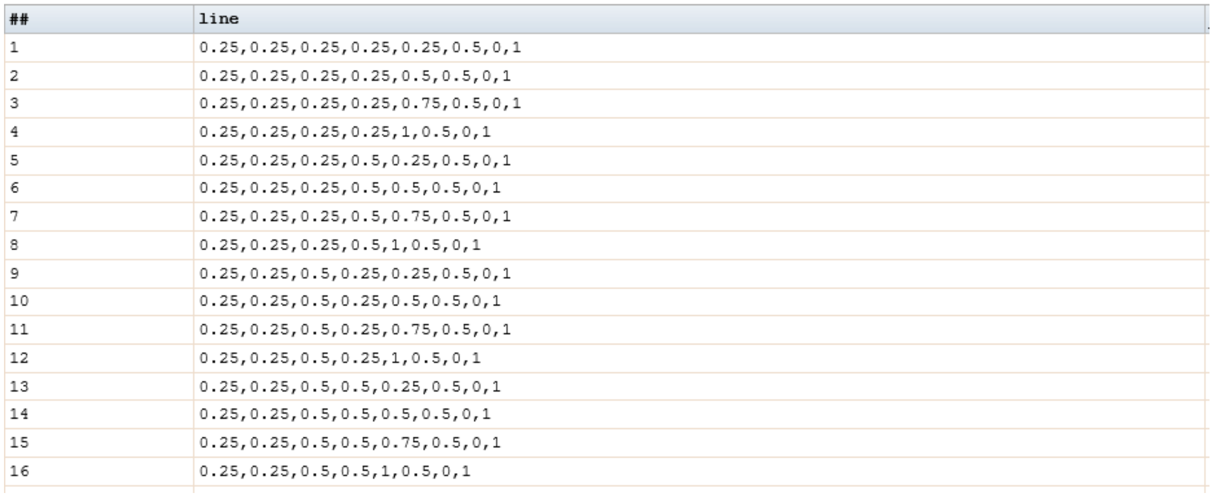
Table 16: Attribute information for shuttle landing control dataset

Number of Attributes : 6
Number of classes : 2
Number of Samples : 278

Attribute Information:

1. Stability
2. Error
3. Sign
4. Wind
5. Magnitude
6. Visibility

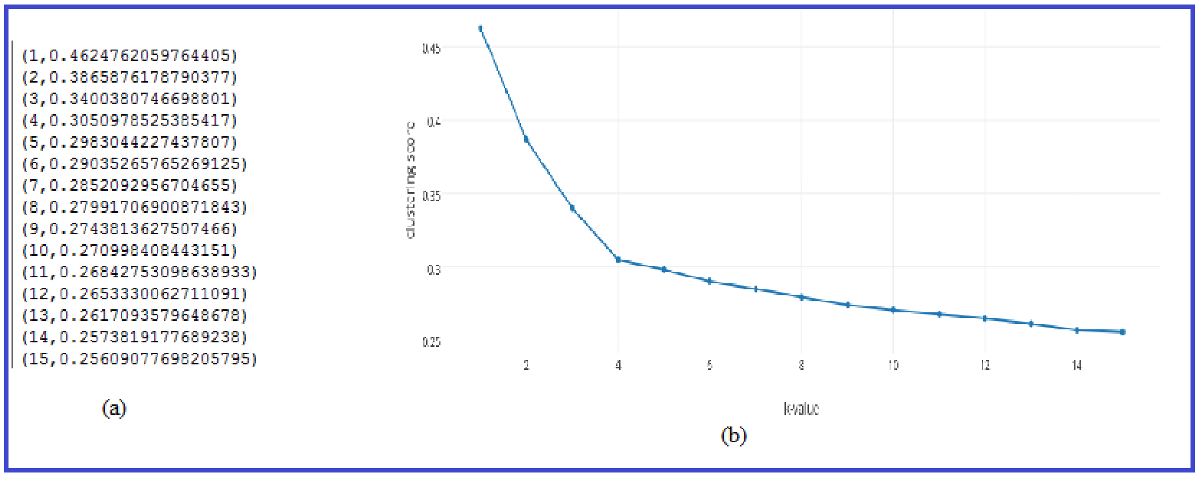
noauto and auto – Classes



##	line
1	0.25,0.25,0.25,0.25,0.25,0.5,0,1
2	0.25,0.25,0.25,0.25,0.5,0.5,0,1
3	0.25,0.25,0.25,0.25,0.75,0.5,0,1
4	0.25,0.25,0.25,0.25,1,0.5,0,1
5	0.25,0.25,0.25,0.5,0.25,0.5,0,1
6	0.25,0.25,0.25,0.5,0.5,0.5,0,1
7	0.25,0.25,0.25,0.5,0.75,0.5,0,1
8	0.25,0.25,0.25,0.5,1,0.5,0,1
9	0.25,0.25,0.5,0.25,0.25,0.5,0,1
10	0.25,0.25,0.5,0.25,0.5,0.5,0,1
11	0.25,0.25,0.5,0.25,0.75,0.5,0,1
12	0.25,0.25,0.5,0.25,1,0.5,0,1
13	0.25,0.25,0.5,0.5,0.25,0.5,0,1
14	0.25,0.25,0.5,0.5,0.5,0.5,0,1
15	0.25,0.25,0.5,0.5,0.75,0.5,0,1
16	0.25,0.25,0.5,0.5,1,0.5,0,1

Figure 28: Snapshot of shuttle landing control dataset

Figure 28 gives a snapshot of how shuttle landing control data looks like. Last two columns are for classes (01 and 10), this is because dataset is normalized.



```
(1,0.4624762059764405)
(2,0.3865876178790377)
(3,0.3400380746698801)
(4,0.3050978525385417)
(5,0.2983044227437807)
(6,0.29035265765269125)
(7,0.2852092956704655)
(8,0.27991706900871843)
(9,0.2743813627507466)
(10,0.270998408443151)
(11,0.26842753098638933)
(12,0.2653330062711091)
(13,0.2617093579648678)
(14,0.2573819177689238)
(15,0.25609077698205795)
```

Figure 29: Clustering scores and its graph for shuttle dataset

Figure 29 (a) refers to clustering score for k-values from 1 to 15

Figure 29 (b) graph plotted for k-values and their clustering scores in Figure 29 (a)

From Figure 29, best k-value is chosen to be 4 because decrease after 4 is not noticeable.

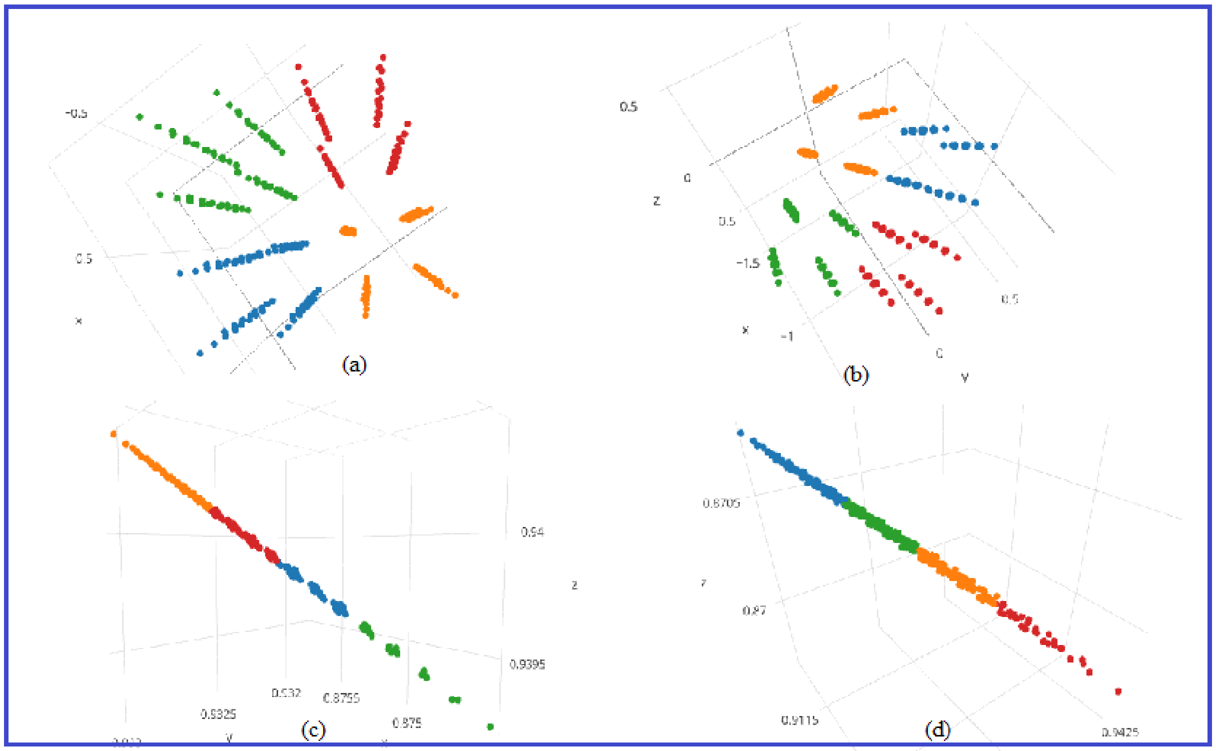


Figure 30: Visualizations for shuttle landing control reduced 3-dimensional dataset

Figure 30 (a) - visualization for PCA reduced dataset

Figure 30 (b) - visualization for SVD reduced dataset

Figure 30 (c) - visualization for DBN reduced dataset

Figure 30 (d) - visualization for stacked auto-encoders reduced dataset

From the visualizations in Figure 30, overlapping is not visible in any case but PCA and SVD have clearly separable classes while others are not distinguishable. Table 17 gives aggregate entropy values for shuttle dataset.

Table 17: Entropy values for shuttle landing dataset

Reduction Technique	**Entropy Value**
PCA	0.0
SVD	0.0
Deep Belief Networks	0.41675653452762

Stacked Auto Encoders	0.3920528957451333

If clearly observed after checking the entropy results, visualization for stacked auto-encoders is somewhat better and overlapping is less when compared to DBN. Of course PCA and SVD again has clear separable classes because of which entropy resulted in 0.

4.6 Dataset 5: Ecoli

This data contains protein localization sites [44]. Table 19 gives attribute information for ecoli dataset.

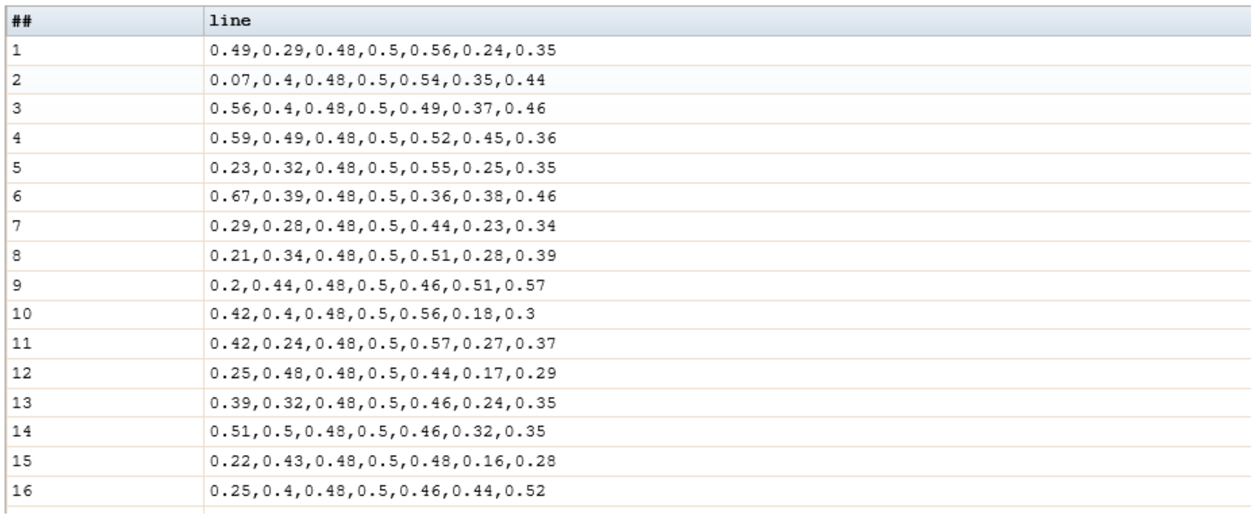
Table 18: Attribute information for ecoli dataset

Number of Attributes : 7
Number of classes : 8
Number of Samples : 336

Attribute Information:

1. mcg
2. gvh
3. lip
4. chg
5. aac
6. alm1
7. alm2

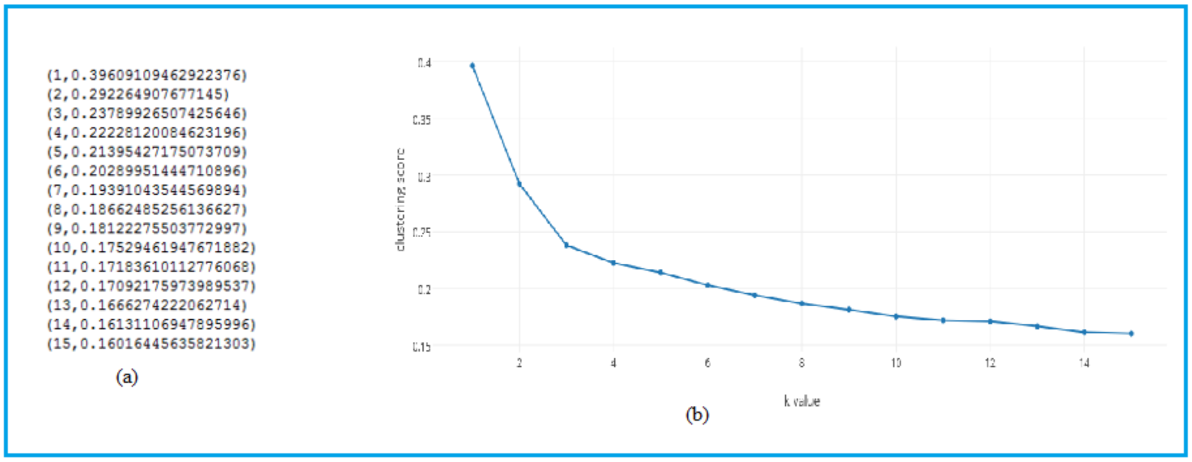
cytoplasm, inner membrane without signal sequence, periplasm, inner membrane uncleavable signal sequence, outer membrane, outer membrane lipoprotein, inner membrane lipoprotein, inner membrane cleavable signal sequence – Classes



##	line
1	0.49,0.29,0.48,0.5,0.56,0.24,0.35
2	0.07,0.4,0.48,0.5,0.54,0.35,0.44
3	0.56,0.4,0.48,0.5,0.49,0.37,0.46
4	0.59,0.49,0.48,0.5,0.52,0.45,0.36
5	0.23,0.32,0.48,0.5,0.55,0.25,0.35
6	0.67,0.39,0.48,0.5,0.36,0.38,0.46
7	0.29,0.28,0.48,0.5,0.44,0.23,0.34
8	0.21,0.34,0.48,0.5,0.51,0.28,0.39
9	0.2,0.44,0.48,0.5,0.46,0.51,0.57
10	0.42,0.4,0.48,0.5,0.56,0.18,0.3
11	0.42,0.24,0.48,0.5,0.57,0.27,0.37
12	0.25,0.48,0.48,0.5,0.44,0.17,0.29
13	0.39,0.32,0.48,0.5,0.46,0.24,0.35
14	0.51,0.5,0.48,0.5,0.46,0.32,0.35
15	0.22,0.43,0.48,0.5,0.48,0.16,0.28
16	0.25,0.4,0.48,0.5,0.46,0.44,0.52

Figure 31: Snapshot for ecoli dataset

Figure 31 shows the original form of ecoli dataset. Since all the data values are in range [0,1], normalization is not performed for this dataset.



```
(1,0.39609109462922376)
(2,0.292264907677145)
(3,0.23789926507425646)
(4,0.22228120084623196)
(5,0.21395427175073709)
(6,0.20289951444710896)
(7,0.19391043544569894)
(8,0.18662485256136627)
(9,0.18122275503772997)
(10,0.17529461947671882)
(11,0.17183610112776068)
(12,0.17092175973989537)
(13,0.1666274222062714)
(14,0.16131106947895996)
(15,0.16016445635821303)
```

Figure 32: Clustering Scores and its graph for ecoli dataset

From Figure 32, it is determined that k-value as 3 works best for ecoli dataset.

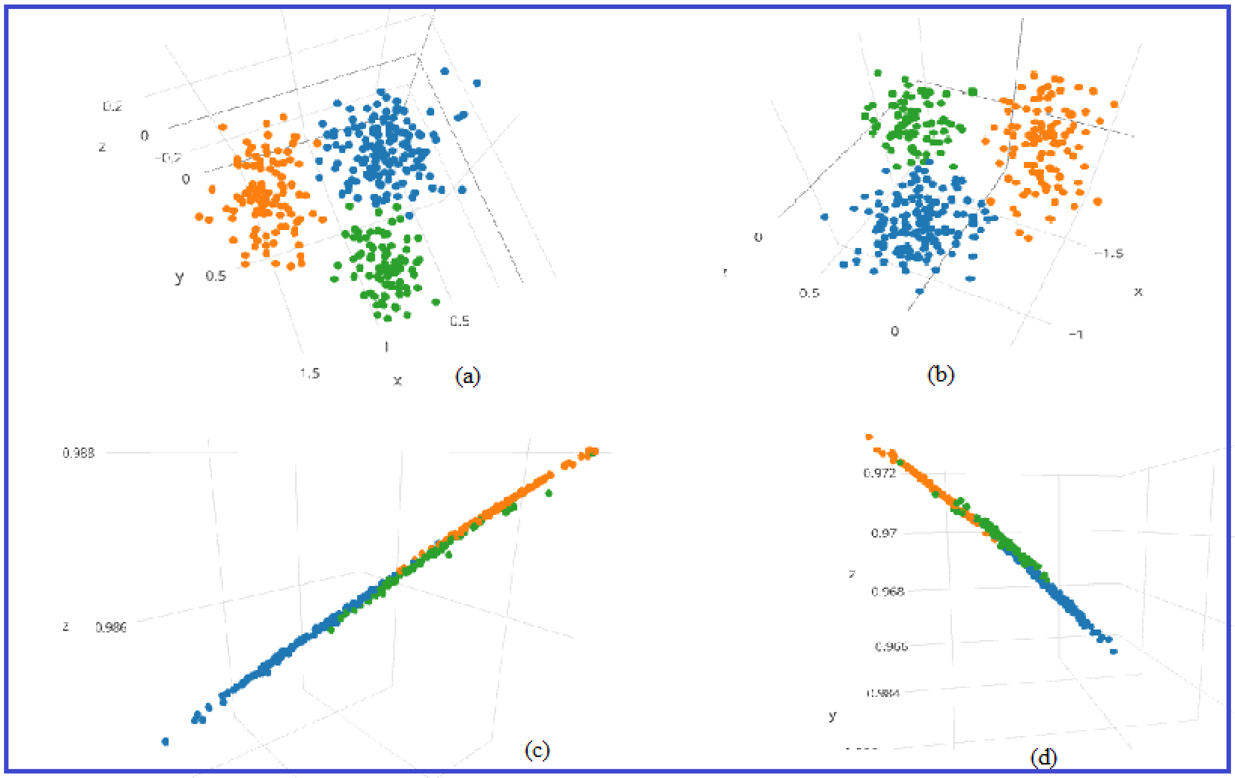


Figure 33: Visualizations for ecoli reduced 3-dimensional dataset

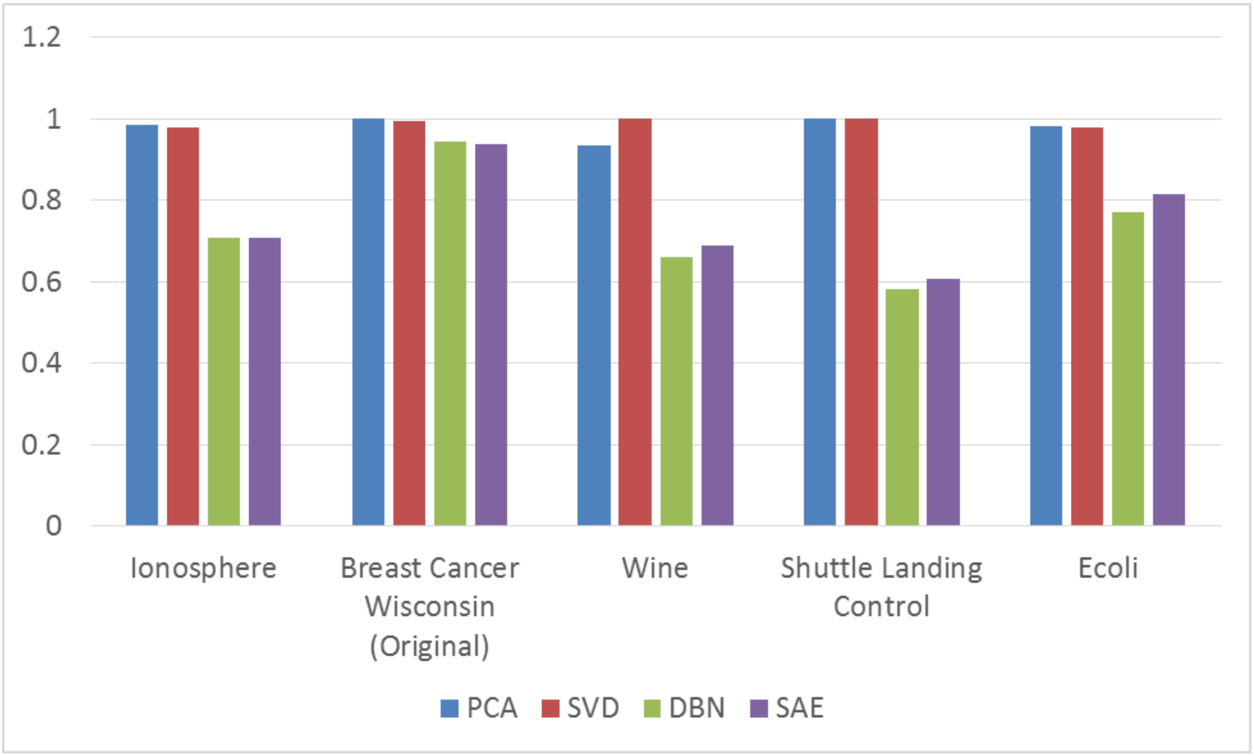
Figure 33 has PCA visualization is more clear than SVD (because few samples in two classes green and blue are so close). There is overlapping in DBN and Stacked auto-encoders. Let us check the aggregate entropy results for this dataset.

Table 19: Entropy values for ecoli dataset

Reduction Technique	**Entropy Value**
PCA	0.01677532448241407
SVD	0.02091038972074466
Deep Belief Networks	0.2302726895610426
Stacked Auto Encoders	0.1859857487446476

Overlapping is not visible in Figure 33, but since the entropy is low only for PCA, classes are separable only in PCA and loss of information is less when compared to other techniques.

Finally, we draw a bar chart with obtained entropy values for each dataset subtracted from 1 in order to give an intuitive view of comparison.



Bar Chart 1: Comparison for all datasets with four dimension reduction techniques

Now we have to check for the highest performance in each of the datasets which are near to 1 (since original entropy value is subtracted from 1). From Bar Chart 1, it is clear that in almost all the datasets, traditional techniques prevail over deep learning techniques and moreover PCA performed much better than SVD.

Thus at the end of this chapter, conclusion can be given as PCA is observed to be the best dimension reduction technique based on the conducted experiments.

CHAPTER 5

Technology Overview and Implementation Part

5.1 Overview

In this Chapter, firstly we introduce HPCC as a platform and its benefits for being used for machine learning and big data, then we demonstrate the implementation algorithm and logic details for deep belief networks. HPCC platform is used for validating the results for all the dimension reduction techniques. HPCC platform has its own machine learning and deep learning modules. All the other techniques SVD, PCA and Stacked auto-encoders are already available in the HPCC deep learning library except Deep Belief Networks. Hence it is decided to implement Deep Belief Networks in HPCC platform and use it for experiments.

5.2 Technology Overview

Figure 34, image is copied from [15], gives complete picture of HPCC machine learning library. HPCC has a lot of modules which are useful for developing deep learning module.

5.3 HPCC Platform

HPCC is a Platform provided by Lexis Nexis organization to meet the data-intensive computing requirements of companies that involve in processing and analyzing massive data. HPCC platform is capable of distributed storage system, job execution environment, online querying, parallel processing, and parallel development and also easy to use.

HPCC architecture comprises of Thor, Roxie and other middle-ware components. Both thor and roxie are referred to as cluster components. Thor is a data refinery, it involves in cleansing the raw data that is sprayed on to it while roxie is a rapid data delivery engine. Roxie tries to search a particular record or a group of records based on indexes. Thor is used to process the data and roxie is used as a flexible storage with indexes. Roxie cluster performance depends on different factors such as machine speed, data complexity, number of nodes and nature of query.

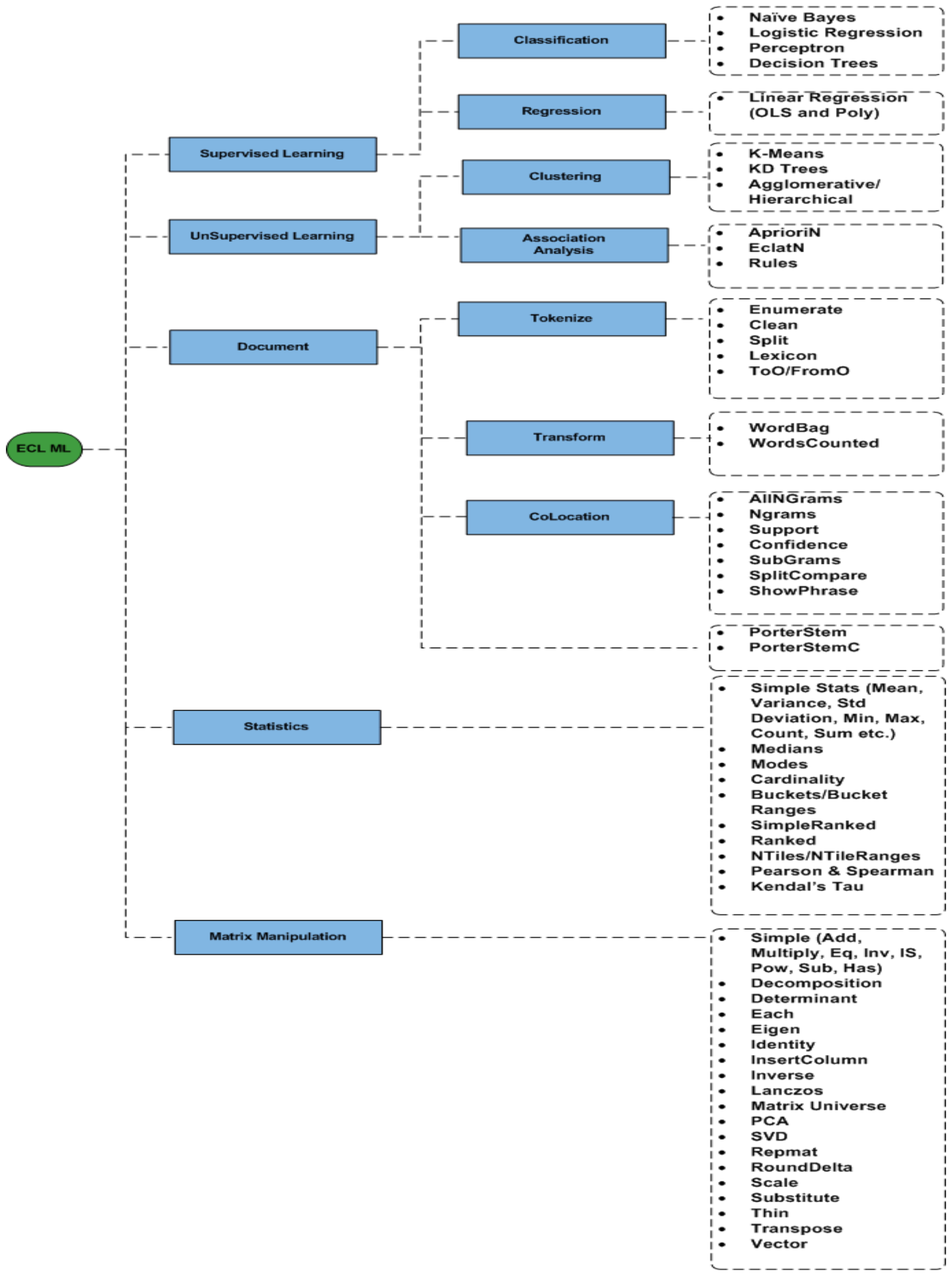


Figure 34: HPCC Machine Learning Hierarchy[15]

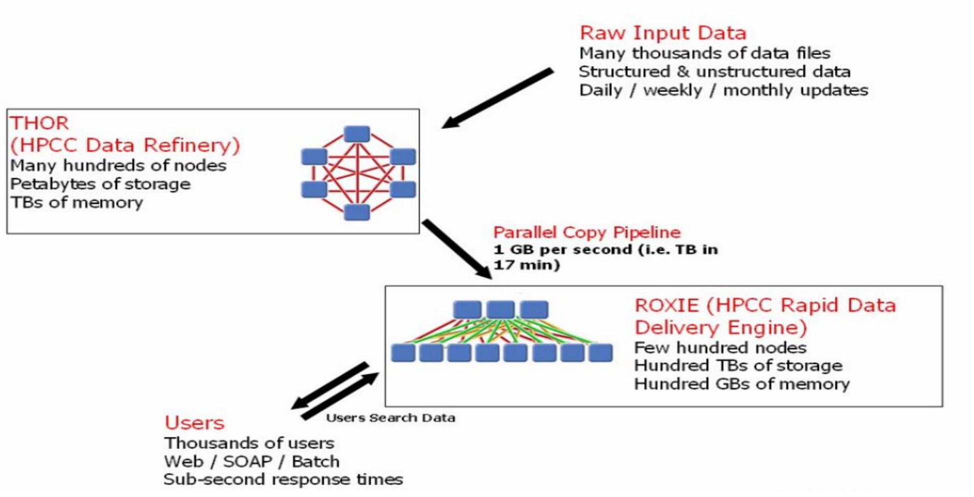


Figure 35: HPCC Architecture [45]

Figure 35 is copied from [45] and shows components of HPCC namely Thor and Roxie.

ECL, Enterprise Control Language, is the key reason for HPCC flexible processing capabilities. It is simple, declarative and highly optimized. The flow of execution does not follow a determined order but it depends on the flow of data and transformations specified. ECL is a case-insensitive language that has familiar syntax. The amount of code required is very less to perform a particular function when compared to other languages like Java and C++. ECL code is compiled and converted into optimized C++ code that finally gets executed on HPCC platform. It is also possible to incorporate C++ functions inline in ECL programs. ECL is extensible for data definition, filtering, data management and data transformation. Figure 36 shows ECL editor and ECL Watch [46], a graphical or web-based user interface for HPCC platform.

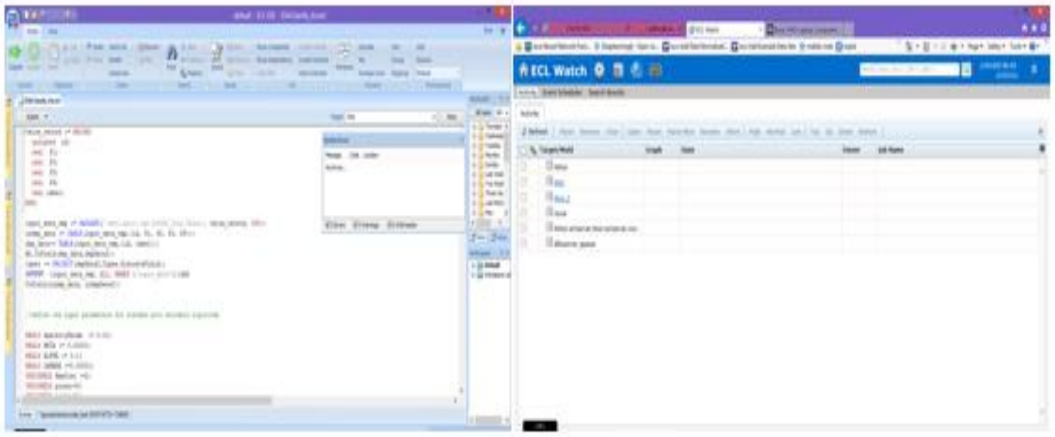


Figure 36: ECL IDE and ECL Watch

Figure 36 has screenshots for ECL IDE (left-part of the figure) and ECL Watch (right-part of the figure).

High performance cluster computing environment with thor, roxie and ECL provides a complete solution for data-intensive applications and big data analytics. Two distinct computing clusters, thor that can process large volumes of data and roxie, that can perform efficient retrieval in less time along with ECL combined together called HPCC platform is chosen for our experiments. Section 5.4 gives an overview of how it differs from other big data solutions like Hadoop and section 5.5 clearly lists the modules that are provided by HPCC which can be specifically used for machine learning.

5.4 HPCC Vs Hadoop

The operating system platform used for both HPCC and Hadoop are Linux and windows. System configurations: HPCC clusters can be implemented in two configurations: Thor (Data Refinery) is comparable to the Hadoop MapReduce and Roxie (Data Delivery Engine) provides online query processing and data warehouse capabilities. HPCC systems has multiple clusters with both configurations. Hadoop system implements clusters with MapReduce processing. The cluster functions as distributed file system running HDFS. Another function which lays on the head of Hadoop MapReduce and HDFS system software includes HBase, Hive, etc [15].

Platform Languages: HPCC uses ECL as a primary programming language as environment and ECL is compiled into C++ which is then compiled into DLLs for the execution of Thor and Roxie platform. Hadoop systems generally uses Java programming language and other languages are implemented using streaming or pipe interface. HBase and Hive have their own language as interface.

Performance: HPCC is 3.95 times faster than Hadoop in certain benchmarks [15]. It is given that HPCC implemented 1TB on 400 node-system in 102 seconds whereas Hadoop took 1 TB on 1460 nodes in 62 seconds [15].

Scalability: HPCC require less nodes for the equal processing as a Hadoop cluster. Hadoop requires one to thousands of nodes.

5.5 HPCC, Machine Learning and Deep Learning

HPCC provides different modules like classification, association, cluster, correlations and discretize to work with. There are also few routines that help in implementing new algorithms or to effectively deal with matrix calculations. PBblas, Mat, DMat are examples for such routines. Mat and DMat libraries give access to various routines for matrix calculations like adding, multiplying, transpose etc…PBblas library is used for computing matrices in parallel blocks. It is the responsibility of the user to give the partitioning blocks correctly for matrices. In general, PBblas library is used to speed up the matrix computations.

5.6 ECL Experience and Challenges

ECL is a data-centric language. It is different from traditional programming languages like C, Java where the problem is dealt from "how-to-solve" perspective but in ECL the problem is approached from "what-to-solve" perspective. ECL programming is not sophisticated, it is a simple declarative language and easy to learn. Data handling is also easier with ECL because of its built-in libraries and features like transformations and aggregations.

Parallel processing and distributed environment is another key feature where ECL stands worthy. The architecture takes care of parallel processing, nothing has to be written in the form of code. At the same time, user can make practice of "DISTRIBUTE" functionality that re-distributes the records across all the nodes of the cluster.

My experience with ECL programming is presented in different phases below:

Phase 1: Testing Using ECL (machine learning algorithms)

Initial experience with ECL started when I tested few of the machine learning algorithms from Figure 34. My job was to execute k-means, decision trees, naïve bayes, linear regression and logistic regression using different big datasets. At this point of time, I did not have much of ECL programming knowledge as the primary task was just to see the performance of existing machine learning algorithms.

ECL Knowledge – Basic Syntax, Using ECL Watch, Executing Programs

Phase 2: Understanding ECL (with Decision Trees)

During this phase, I got close to ECL programming as there is need to understand the implementation of an algorithm named "decision trees". I started learning ECL and also use it practically. Learning part was easier but applying ECL was bit difficult because problem has to be approached from a different perspective in ECL.

ECL Knowledge – Problem Solving with ECL, Writing Queries, Using TRANSFORM, PROJECT, TABLE, JOINs

Phase 3: Enhancement using ECL (PageRank Algorithm)

This phase is a start to development though not pure development. An existing implementation of page rank algorithm is modified to remove redundant code and make it readable. There is repetitive code for iterations part of the algorithm, it has been removed by making using of "LOOP" structure in ECL. The code also has been enhanced to add threshold feature.

ECL Knowledge – LOOP, MODULE, FUNCTION and many other structures in ECL

Phase 4: Deep Learning using ECL (PBblas module)

This is a step to get some knowledge on existing libraries for machine learning especially to deal with matrices. Since parallel block implementation of matrices is very much required for big data.

ECL Knowledge – ML.Mat Library, ML.DMat Library, PBblas Library

Phase 5: Development in ECL (Deep Belief Networks)

Here I start with pure development of a deep learning model called "Deep Belief Networks" building it from the scratch in ECL. More details about the implementation, results, validation, testing for the algorithm is provided in section 5.7.

Following are few of the challenges encountered during the development process in ECL:

- Nesting loops: Graphs created from nested loops look very different from normal loops.
- Understanding Errors: ECL compiler errors are sometimes difficult to understand and it takes time to point out the error correctly.
- Runtime Issues: ECL compiler converts each of the statements written in ECL to objective C++ code internally, during this process of conversion few of the unused ECL statements are ignored. For example, let us assume there is a check for matrix dimensions which can be known only during run time. Though in the program, matrix dimensions are incompatible, ECL will not give any error if that particular result is never being used or shown to the user as an output.
- Module Exports: This is a kind of return statement from module because we do not have explicit return in module. But it is real difficult to implement the logic when there are multiple values to be returned from the module.
- Querying using GROUP: The concept of GROUP in ECL is similar to GROUP BY clause in SQL. GROUP can be used only with aggregate functions. Using GROUP and aggregate functions in record structures is slightly confusing.

Phase 6: More development in ECL

As part of the course curriculum, implemented basic version of logistic regression, multi-layer perceptron and multi-layer perceptron with contrastive divergence weight updating technique. Apart from these, I have improved my ECL knowledge by assisting the class and delivering the lectures on "Deep Learning in ECL".

5.7 Deep Belief Network Implementation

Deep Belief Network is a stack of Restricted Boltzmann Machines. In order to implement deep belief network, the key is to implement its significant block Restricted Boltzmann Machine.

Table 20: Deep Belief Networks Algorithm

"

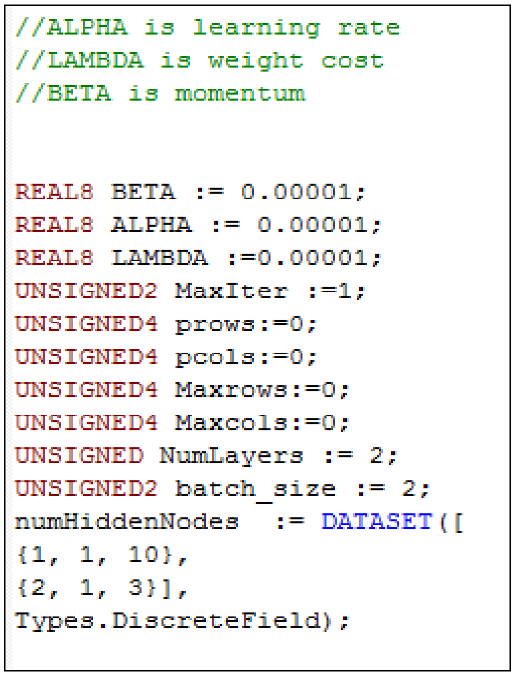
1. Train the first layer as an RBM that models the raw input x = h(0) as its visible layer.
2. Use that first layer to obtain a representation of the input that will be used as data for the second layer. Two common solutions exist. This representation can be chosen as being the mean activations P(h(1) = 1|h(0)) or samples of P(h(1)|h(0)).
3. Train the second layer as an RBM, taking the transformed data (samples or mean activations) as training examples (for the visible layer of that RBM).
4. Iterate (2 and 3) for the desired number of layers, each time propagating upward either samples or mean values.
5. Fine-tune all the parameters of this deep architecture with respect to a proxy for the DBN log- likelihood, or with respect to a supervised training criterion (after adding extra learning machinery to convert the learned representation into supervised predictions, e.g. a linear classifier).

"[16]

Table 20 gives the insight of deep belief networks algorithm implementation. It is based on Hinton's practical guide in [47] and the algorithm is from [16].

Different parameters used in the implementation are learning rate, momentum and weight cost. Learning rate cannot be too big as there are chances for reconstruction error to maximize which results in large values for weights. So it is always better to choose learning rate in the range of 0 to 0.5, though it always depends on the data. Momentum increases the learning speed, so it is required to have greater momentum during initial few iterations but then it's always better to decrease the momentum. In this manner, learning can be very stable. Weight cost is an additional term added to the normal gradient. The weight cost best works if it is in the range of 0.01 to 0.00001.

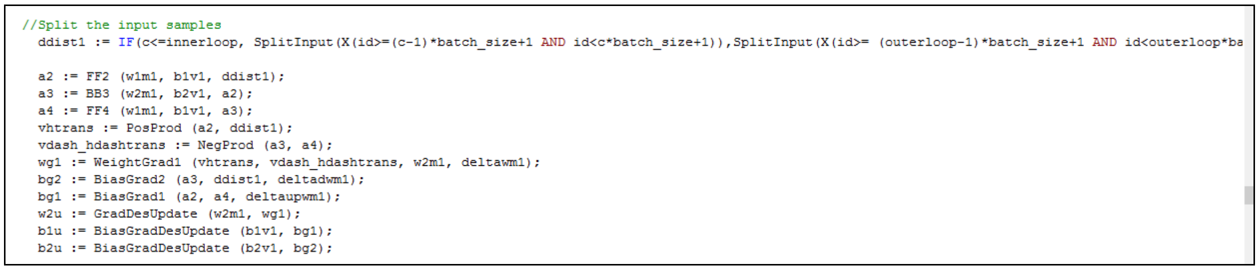
Mini-batch gradient descent technique is also used for implementation which is from [16, 47]. That means dataset is not fed to the network as a whole at a time. Instead algorithm in Table 20 is repeated for each mini batch of the dataset. “batch_size” is a user-defined parameter, where user can specify number of batches in which dataset has to be processed.



```
//ALPHA is learning rate
//LAMBDA is weight cost
//BETA is momentum


REAL8 BETA := 0.00001;
REAL8 ALPHA := 0.00001;
REAL8 LAMBDA :=0.00001;
UNSIGNED2 MaxIter :=1;
UNSIGNED4 prows:=0;
UNSIGNED4 pcols:=0;
UNSIGNED4 Maxrows:=0;
UNSIGNED4 Maxcols:=0;
UNSIGNED NumLayers := 2;
UNSIGNED2 batch_size := 2;
numHiddenNodes  := DATASET([
{1, 1, 10},
{2, 1, 3}],
Types.DiscreteField);
```

Figure 37: Parameters for DBN

Figure 37 gives an overview of all parameters being passed to deep belief networks module. NumLayers specify the layers in the network architecture chosen for the dataset. numHiddenNodes also relate to the network architecture.



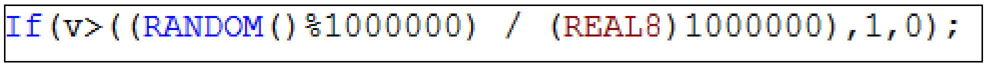
```
//Split the input samples
  ddist1 := IF(c<=innerloop, SplitInput(X(id>=(c-1)*batch_size+1 AND id<c*batch_size+1)),SplitInput(X(id>= (outerloop-1)*batch_size+1 AND id<outerloop*ba

  a2 := FF2 (w1m1, b1v1, ddist1);
  a3 := BB3 (w2m1, b2v1, a2);
  a4 := FF4 (w1m1, b1v1, a3);
  vhtrans := PosProd (a2, ddist1);
  vdash_hdashtrans := NegProd (a3, a4);
  wg1 := WeightGrad1 (vhtrans, vdash_hdashtrans, w2m1, deltawm1);
  bg2 := BiasGrad2 (a3, ddist1, deltadwm1);
  bg1 := BiasGrad1 (a2, a4, deltaupwm1);
  w2u := GradDesUpdate (w2m1, wg1);
  b1u := BiasGradDesUpdate (b1v1, bg1);
  b2u := BiasGradDesUpdate (b2v1, bg2);
```

Figure 38: Code Snippet

The code shown in Figure 38 is the most essential and central part of deep belief networks. It talks about contrastive divergence and gradient updates.

Output can be learnt model and the final result from DBN. If there are ‘n’ neurons in the output layer of DBN, then there are ‘n’ dimensions or features in the final result of DBN. Thus dimension reduction in DBN can be performed just by giving the reduced network architecture.

There is a little variation in the implementation of Deep Belief Networks for binary version. In continuous version, the output obtained in positive and negative phases is used in the same manner. That is output at any layer is a set of probabilities. But in binary version, during the positive phase, we consider hidden states instead of hidden probabilities. The update from probabilities to states is performed stochastically to avoid any unnecessary sampling noise [47]. To make it clear, a quick snippet of code is given in Figure 39.

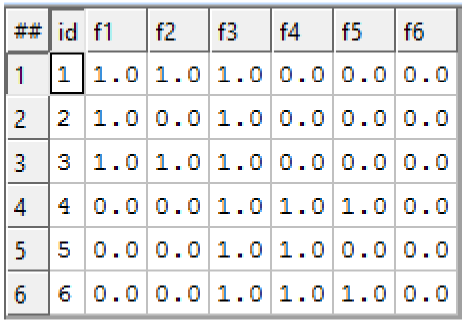


```
If(v>((RANDOM()%1000000) / (REAL8)1000000),1,0);
```

Figure 39: Code Snippet

‘v’ is a probability at hidden layer, then except final update to hidden layer, all the other updates go through stochastic updating of hidden probabilities to hidden states. Stochastic updating means for example ‘v’ is compared with some randomly generated probability and if ‘v’ is greater than randomly generated probability, then state is updated as ‘1’ or else state is updated as 0. It is crucial for the hidden layer neurons to return states to visible layer neurons rather than probabilities because if probabilities are used, then there may be information bottle neck. Since there is no communication to the visible layer after the final hidden update, there won’t be any problem if stochastic updating is not used for the final hidden update.

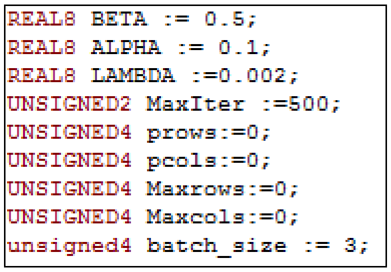
Figure 40-44 give input, intermediate results, output for movie-ratings, a simple example dataset executed using implemented HPCC binary version of deep belief networks. All the input data, output data is binary but the intermediate updates at the visible layer have probabilities.



##	id	f1	f2	f3	f4	f5	f6
1	1	1.0	1.0	1.0	0.0	0.0	0.0
2	2	1.0	0.0	1.0	0.0	0.0	0.0
3	3	1.0	1.0	1.0	0.0	0.0	0.0
4	4	0.0	0.0	1.0	1.0	1.0	0.0
5	5	0.0	0.0	1.0	1.0	0.0	0.0
6	6	0.0	0.0	1.0	1.0	1.0	0.0

Figure 40: Movie-Ratings dataset

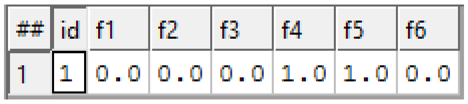
Figure 40 gives all the samples for movie-ratings dataset. All the features or dimensions here are ratings given by a user for different movies. More clearly and verbatically, f1 to f6 are movies, 1 to 6 (id column) are users. Values for f1-f6 are the ratings given by users 1 to 6. User 1 liked movies f1, f2, f3 and did not like the movies f4, f5, f6. Overall only input ratings are obtained from 6 users which is used for feeding the network.



```
REAL8 BETA := 0.5;
REAL8 ALPHA := 0.1;
REAL8 LAMBDA :=0.002;
UNSIGNED2 MaxIter :=500;
UNSIGNED4 prows:=0;
UNSIGNED4 pcols:=0;
UNSIGNED4 Maxrows:=0;
UNSIGNED4 Maxcols:=0;
unsigned4 batch_size := 3;
```

Figure 41: Parameters used for Movie-Ratings dataset

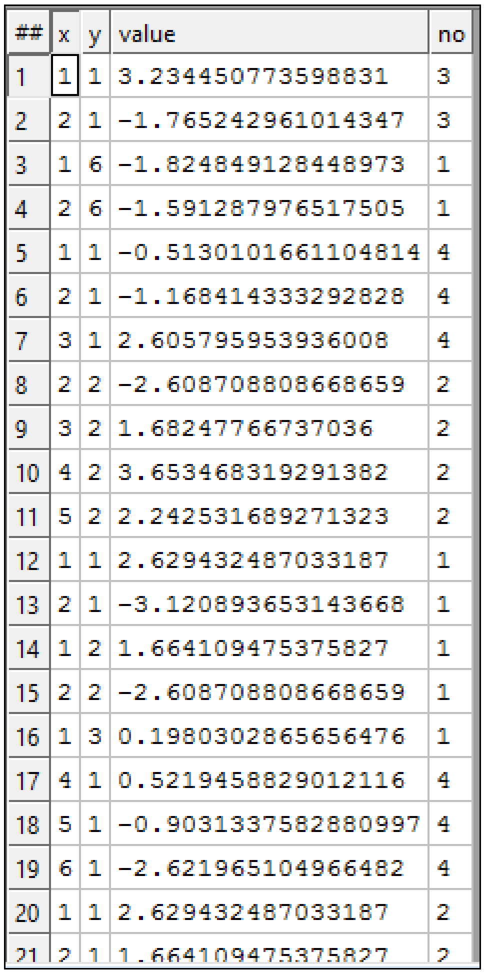
Since the dataset is very small, number of iterations used is very large. Due to high number of iterations, learning converges and we get a constant result every time. Since the batch_size is given as 3, everytime 3 records are processed. Basically for movie-ratings dataset, 2 batches of inputs are given for 500 times.



##	id	f1	f2	f3	f4	f5	f6
1	1	0.0	0.0	0.0	1.0	1.0	0.0

Figure 42: Test data for movie-ratings

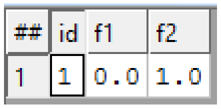
Figure 40 is a training data for the network. When the network learns its model from the inputs given, it is tested for the correctness using test data in Figure 42. We want to check which output neurons are on and which are off and whether the behavior is constant even running it for multiple times. Test data contains only one input i.e, ratings given by just one user for 6 movies.



##	x	y	value	no
1	1	1	3.234450773598831	3
2	2	1	-1.765242961014347	3
3	1	6	-1.824849128448973	1
4	2	6	-1.591287976517505	1
5	1	1	-0.5130101661104814	4
6	2	1	-1.168414333292828	4
7	3	1	2.605795953936008	4
8	2	2	-2.608708808668659	2
9	3	2	1.68247766737036	2
10	4	2	3.653468319291382	2
11	5	2	2.242531689271323	2
12	1	1	2.629432487033187	1
13	2	1	-3.120893653143668	1
14	1	2	1.664109475375827	1
15	2	2	-2.608708808668659	1
16	1	3	0.1980302865656476	1
17	4	1	0.5219458829012116	4
18	5	1	-0.9031337582880997	4
19	6	1	-2.621965104966482	4
20	1	1	2.629432487033187	2
21	2	1	1.664109475375827	2

Figure 43: Learnt Model

Learnt Model consists of set of weights, set of bias for the network architecture. Figure 43 shows only part of the model obtained for movie-ratings dataset.



##	id	f1	f2
1	1	0.0	1.0

Figure 44: Final Output

Figure 44 shows the final output for the test data. It can be interpreted as there are two neurons in the output layer, talking verbatically, there may be two categories of movies. When the test data is fed to the network, always second neuron i.e, second category of movie is turned on. Though the execution is done multiple times, output remains same. Always second neuron is turned on for the given test data. This can happen only when the network is trained sufficiently and correctly. Due to the converging of weights after 500 iterations, learnt model is same everytime and output is fixed for the same training and test data.

This chapter clearly explains the HPCC platform, algorithm used for deep belief networks and illustrates the output using an example dataset.

In the next Chapter, we present the conclusion and future work.

CHAPTER 6

Conclusions and Future Work

6.1 Conclusions

Though deep learning is the most powerful way of learning in the area of machine learning and has wide range of applications like image classification, pattern recognition, computer vision etc..., dimension reduction for chosen datasets using deep learning did not prove to be good when compared to traditional techniques. Dimension reduction facilitates better visualization of data and easy for analysis. Four techniques for dimension reduction are tested for varied datasets and entropy score is used as the evaluation criteria. Our proposed approach of using k-means clustering, assigning new cluster labels, visualization and finding the entropy to check the data loss is new and easy to evaluate the results.

From the extensive survey we have done (Chapter 2), we find that most literature works use reconstruction error or mean squared error, using the original labels for finding the error after reduction. But we find that the proposed approach works good in most of the cases, helps in giving the loss of information after the dimension reduction. Since deep learning is the recent break through, there is not much work comparing traditional techniques with deep learning approaches. Very few studies have this comparison done but not comprehensive. Many works had comparison work done between linear and non-linear techniques for dimension reduction.

We have proposed the clear approach for dimension reduction and evaluation in Chapter 3. The approach is straight forward, applied for datasets and got promising results. Clustering the data using k-means plays crucial role in proposed approach. Instead of original labels, cluster labels are used and entropy is determined. The process mentioned has to be repeated for all the dimension reduction techniques and entropy score is compared. One with lowest entropy score is decided to be the best suitable dimension reduction technique for the dataset tested.

Chapter 4 illustrates the application of the proposed approach to the datasets like Ionosphere, Breast Cancer Wisconsin (Original), Wine, Shuttle Landing Control and Ecoli. For all the datasets, PCA had the lowest entropy score and better visualization when compared to other techniques. Thus PCA prevailed to be the best technique for dimension reduction.

Chapter 6 discusses the HPCC technology, using HPCC as a platform, implementation of deep belief networks algorithm in HPCC deep learning module. There are two versions of deep belief networks, one for continuous data and other for binary data. All the case studies in chapter 4 are tested using deep belief networks continuous version. Hence a simple movie-ratings dataset is used to validate the results of binary version.

With the proposed approach, results show that traditional techniques performed better when compared to deep learning techniques.

6.2 Future Work

Though 2D or 3D space is decent for visualization purpose, it is not always true that a dataset will have good classification results or will retain structure of the data with reduced dimensions. Sometimes depending on the nature of data, even higher dimensional space (higher than the original dimensional space) works well. Our future work includes mapping data automatically to different dimensional spaces by using deep learning techniques and then through cross validation model, select the best performed model as the final output. This way, it is easy to know whether reduction or expansion of dimensions is suitable for the given data.

Another direction would be building a deep learning architecture using PCA since PCA proved to have better results over all other techniques. Hence an architecture is proposed to check the layer-by-layer results with PCA, calculate the errors and learn the weights, bias instead of assigning random weights, bias in a network.

Appendix A: Source Code

Deep Belief Networks

```
IMPORT ML;
IMPORT * FROM $;
IMPORT $.Mat;
IMPORT * FROM ML.Types;
IMPORT PBblas;

Layout_Part := PBblas.Types.Layout_Part;
Layout_Cell := PBblas.Types.Layout_Cell;

EXPORT DBN_N := MODULE
EXPORT DBN_IntWeights (INTEGER4 NumberofFeatures, INTEGER4
NumberofHiddenLayerNodes) := FUNCTION
  net := DATASET([
  {1,1, NumberofFeatures},
  {2,1,NumberofHiddenLayerNodes},
  {3,1,NumberofFeatures}],
  Types.DiscreteField);
  RETURN NeuralNetworks(net).IntWeightsDBN;
END;
EXPORT DBN_IntBias (INTEGER4 NumberofFeatures, INTEGER4
NumberofHiddenLayerNodes) := FUNCTION
  net := DATASET([
  {1, 1, NumberofFeatures},
  {2,1,NumberofHiddenLayerNodes},
  {3,1,NumberofFeatures}],
  Types.DiscreteField);
  RETURN NeuralNetworks(net).IntBias;
END;

/*
//Implementation of the Restricted Boltzmann Machine
//beta: momentum
//IntW : initial weights for the Restricted Boltzmann Machine
//IntW includes two matrices of size Number_of_hidden_layer_nodes *
Number_of_features and the transpose of the weight matrix
(Number_of_features * Number_of_hidden_layer_nodes)
//IntB : Initial Bias for the Restricted Boltzmann Machine
//IntB includes two matrices of size Number_of_hidden_layer_nodes*1
and Number_of_features*1
//LAMBDA : weight cost term, costs of weight update
//ALPHA : learning rate
//MaxIter : Maximum number of iterations
//batch_size : to number of batches given input samples need to be
splitted
//prows, pcols, Maxrows, Maxcols for the Pbblas partitioning:
```

```
// - prows: an optional parameter used to set the number of rows in
partition blocks (Should be used in conjuction with pcols)
// - pcols: an optional parameter used to set the number of cols in
partition blocks (Should be used in conjuction with prows)
// - Maxrows: an optional parameter used to set maximum rows allowed per
block when using AutoBVMap
// - Maxcols: an optional parameter used to set maximum cols allowed per
block when using AutoBVMap
*/
EXPORT RBM_Build (UNSIGNED4 prows=0, UNSIGNED4 pcols=0, UNSIGNED4
Maxrows=0, UNSIGNED4 Maxcols=0) := MODULE
  //this is a un-supervised learning algorithm, no need for the labled
data
  SHARED RBM(DATASET(Types.NumericField) X, DATASET(Mat.Types.MUElement)
IntW, DATASET(Mat.Types.MUElement) Intb, REAL8 BETA=0.5, REAL8
LAMBDA=0.0002, REAL8 ALPHA=0.1, UNSIGNED2 MaxIter=100,UNSIGNED2
batch_size=10) := MODULE
    // it is same as indep datac represented as matrix
            dt := Types.ToMatrix(X);
            dTmp := dt;
            SHARED d := Mat.Trans(dTmp); //in the entire of the
calculations we work with the d matrix that each sample is presented in
one column
            SHARED m := batch_size;
            SHARED m_1 := 1/m; // 1/6
            SHARED sizeRec := RECORD
                  PBblas.Types.dimension_t m_rows;
                  PBblas.Types.dimension_t m_cols;
                  PBblas.Types.dimension_t f_b_rows;
                  PBblas.Types.dimension_t f_b_cols;
            END;
   //Map for Matrix d.
            SHARED havemaxrow := maxrows > 0;
            SHARED havemaxcol := maxcols > 0;
            SHARED havemaxrowcol := havemaxrow and havemaxcol;
            SHARED dstats := Mat.Has(d).Stats;         // It is a table
with number of elements as 18, xmax as 3 and ymax as 6
            SHARED d_n := dstats.XMax;
            SHARED d_m := batch_size;

            derivemap := IF(havemaxrowcol, PBblas.AutoBVMap(d_n,
d_m,prows,pcols,maxrows, maxcols),
                   IF(havemaxrow, PBblas.AutoBVMap(d_n,
d_m,prows,pcols,maxrows),
                      IF(havemaxcol, PBblas.AutoBVMap(d_n,
d_m,prows,pcols,,maxcols),
                      PBblas.AutoBVMap(d_n, d_m,prows,pcols))));
             SHARED sizeTable :=
 DATASET([{derivemap.matrix_rows,derivemap.matrix_cols,derivemap.part_rows
 (1),derivemap.part_cols(1)}], sizeRec);

             //Create block matrix d
             dmap :=
 PBblas.Matrix_Map(sizeTable[1].m_rows,sizeTable[1].m_cols,sizeTable[1].f_
 b_rows,sizeTable[1].f_b_cols);
             ddist := DMAT.Converted.FromElement(d,dmap);
             //Create block matrix Ytmp
             Ymap := dmap;
             Ydist := ddist;

     //=====================================Function added for splitting
 data===============================================

             SplitInput(DATASET(Types.NumericField) X) := FUNCTION

                   dt := Types.ToMatrix(X);
                   dTmp := dt;
                   dsplit := Mat.Trans(dTmp);
                   msplit := MAX (dsplit, dsplit.y); //number of samples
 is 6
                   msplit_1 := 1/msplit; // 1/6

                   splitSizeRec := RECORD
                         PBblas.Types.dimension_t m_rows;
                         PBblas.Types.dimension_t m_cols;
                         PBblas.Types.dimension_t f_b_rows;
                         PBblas.Types.dimension_t f_b_cols;
                   END;
             //Map for Matrix d.
                   havemaxrowsplit := maxrows > 0;
                   havemaxcolsplit := maxcols > 0;
                   havemaxrowcolsplit := havemaxrowsplit and
 havemaxcolsplit;
                   dsplitstats := Mat.Has(d).Stats;         // It is a
 table with number of elements as 18, xmax as 3 and ymax as 6
                   dsplit_n := dsplitstats.XMax;  //3
                   dsplit_m := dsplitstats.YMax;  //6

                   derivemap := IF(havemaxrowcolsplit,
 PBblas.AutoBVMap(dsplit_n, dsplit_m,prows,pcols,maxrows, maxcols),
                    IF(havemaxrowsplit, PBblas.AutoBVMap(dsplit_n,
 dsplit_m,prows,pcols,maxrows),
                       IF(havemaxcolsplit, PBblas.AutoBVMap(dsplit_n,
 dsplit_m,prows,pcols,,maxcols),
                       PBblas.AutoBVMap(d_n, d_m,prows,pcols))));
                   splitSizeTable :=
 DATASET([{derivemap.matrix_rows,derivemap.matrix_cols,derivemap.part_rows
 (1),derivemap.part_cols(1)}], splitSizeRec);
             //Create block matrix d
                   dsplitmap :=
 PBblas.Matrix_Map(splitSizeTable[1].m_rows,splitSizeTable[1].m_cols,split
 SizeTable[1].f_b_rows,splitSizeTable[1].f_b_cols);
                   dsplitdist :=
 DMAT.Converted.FromElement(dsplit,dsplitmap);

                   return dsplitdist;
             END;

     //Create block matrices for weights
             w1_mat := Mat.MU.From(IntW,1);  // getting first weight
 matrix from layer 1 to layer 2 in forward direction
             w1_mat_x := Mat.Has(w1_mat).Stats.Xmax;
             w1_mat_y := Mat.Has(w1_mat).Stats.Ymax;
             w1map := PBblas.Matrix_Map(w1_mat_x, w1_mat_y,
 sizeTable[1].f_b_rows, sizeTable[1].f_b_rows);
             w1dist := DMAT.Converted.FromElement(w1_mat,w1map);

             w2_mat := Mat.MU.From(IntW,2); // getting second weight
 matrix from layer 2 to layer 1 in backward direction
             w2_mat_x := w1_mat_y;
             w2_mat_y := w1_mat_x;
             w2map := PBblas.Matrix_Map(w2_mat_x, w2_mat_y,
 sizeTable[1].f_b_rows, sizeTable[1].f_b_rows);
             w2dist := DMAT.Converted.FromElement(w2_mat,w2map);

     //each bias vector is converted to block format
             b1vec := Mat.MU.From(Intb,1); // bias matrix from layer 1 to
 layer 2
             b1vec_x := Mat.Has(b1vec).Stats.Xmax;
             b1vecmap := PBblas.Matrix_Map(b1vec_x, 1,
 sizeTable[1].f_b_rows, 1);
             b1vecdist := DMAT.Converted.FromElement(b1vec,b1vecmap);

             b2vec := Mat.MU.From(Intb,2); // bias matrix from layer 2 to
 layer 1
             b2vec_x := Mat.Has(b2vec).Stats.Xmax;
             b2vecmap := PBblas.Matrix_Map(b2vec_x, 1,
 sizeTable[1].f_b_rows, 1);
             b2vecdist := DMAT.Converted.FromElement(b2vec,b2vecmap);

             //functions used
             PBblas.Types.value_t siggrad(PBblas.Types.value_t v,
       PBblas.Types.dimension_t r, PBblas.Types.dimension_t c) := v*(1.0-
       v);
```

```
                  PBblas.Types.value_t sigmoid(PBblas.Types.value_t
      v, PBblas.Types.dimension_t r, PBblas.Types.dimension_t c) :=
      1/(1+exp(-1*v));
                  PBblas.Types.value_t
      applybeta(PBblas.Types.value_t v,PBblas.Types.dimension_t
      r,PBblas.Types.dimension_t c) := BETA*v;
                  PBblas.Types.value_t
      applygamma(PBblas.Types.value_t v,PBblas.Types.dimension_t
      r,PBblas.Types.dimension_t c) := -(LAMBDA*ALPHA/m)*v;

              //maps used
                  b1map := PBblas.Matrix_Map(b1vec_x, m,
      sizeTable[1].f_b_rows, sizeTable[1].f_b_cols);
                  b2map := PBblas.Matrix_Map(b2vec_x, m,
      sizeTable[1].f_b_rows, sizeTable[1].f_b_cols);
                  a2map := b1map;
                  a3map := b2map;
                  a4map := b1map;

              //onevec for calculating the yhat
                  Ones_VecMap := PBblas.Matrix_Map(m, 1,
      sizeTable[1].f_b_cols, 1);
              //New Vector Generator with 1's
                  Layout_Cell gen(UNSIGNED4 c, UNSIGNED4 NumRows)
      := TRANSFORM
                        SELF.x := ((c-1) % NumRows) + 1;
                        SELF.y := ((c-1) DIV NumRows) + 1;
                        SELF.v := 1;
                  END;
                //New Vector Generator with 0's
                  Layout_Cell genzero(UNSIGNED4 c, UNSIGNED4
      NumRows) := TRANSFORM
                        SELF.x := ((c-1) % NumRows) + 1;
                        SELF.y := ((c-1) DIV NumRows) + 1;
                        SELF.v := 0;
                  END;

    //Create Ones Vector for the calculations in the step fucntion
            Ones_Vec := DATASET(m, gen(COUNTER, m),DISTRIBUTED);
            Ones_Vecdist := DMAT.Converted.FromCells(Ones_VecMap,
Ones_Vec);

            Zeroes_Vec := DATASET(w2_mat_x*w2_mat_y, genzero(COUNTER,
w2_mat_x*w2_mat_y),DISTRIBUTED);
            Zeroes_Vecdist := DMAT.Converted.FromCells(w2map,
Zeroes_Vec);

            Zeroes1_Vec := DATASET(b1vec_x, genzero(COUNTER,
b1vec_x),DISTRIBUTED);
            Zeroes1_Vecdist  := DMAT.Converted.FromCells(b1vecmap,
Zeroes1_Vec);
```

```
            Zeroes2_Vec := DATASET(b2vec_x, genzero(COUNTER,
b2vec_x),DISTRIBUTED);
            Zeroes2_Vecdist  := DMAT.Converted.FromCells(b2vecmap,
Zeroes2_Vec);


      //Positive Phase involves generating hidden layer samples
      FF2(DATASET(Layout_Part) w1, DATASET(Layout_Part) b1v,
DATASET(Layout_Part) ddist1):= FUNCTION
            b1m := PBblas.PB_dgemm(FALSE, TRUE, 1.0,b1vecmap, b1v,
Ones_VecMap, Ones_Vecdist, b1map);
      //z2 = w1*X+b1;
            z2 := PBblas.PB_dgemm(FALSE, FALSE, 1.0,w1map, w1, dmap,
ddist1, b1map, b1m, 1.0);
      //a2 = sigmoid (z2);
            a2 := PBblas.Apply2Elements(b1map, z2, sigmoid);
            RETURN a2;
      END;//END FF2

      //Positive phase involves generating visible samples given hidden
samples
      BB3(DATASET(Layout_Part) w2,DATASET(Layout_Part) b2v,
DATASET(Layout_Part) a2 ):= FUNCTION
            b2m := PBblas.PB_dgemm(FALSE, TRUE, 1.0,b2vecmap, b2v,
Ones_VecMap, Ones_Vecdist, b2map);
      //z3 = w2*a2+b2;
            z3_tmp := PBblas.PB_dgemm(FALSE, FALSE,1.0,w2map, w2, a2map,
a2, b2map);
            z3 := PBblas.PB_daxpy(1.0, z3_tmp, b2m);
      //a3 := sigmoid (z3)
            a3 := PBblas.Apply2Elements(b2map, z3, sigmoid);
            RETURN a3;
      END;//END BB3

      //Negative phase involves generating hidden layer samples
      FF4(DATASET(Layout_Part) w1, DATASET(Layout_Part) b1v,
DATASET(Layout_Part) a3):= FUNCTION
            b1m := PBblas.PB_dgemm(FALSE, TRUE, 1.0,b1vecmap, b1v,
Ones_VecMap, Ones_Vecdist, b1map);
      //z4 = w1*a3+b1
            z4_tmp := PBblas.PB_dgemm(FALSE, FALSE,1.0,w1map, w1, a3map,
a3, b1map);
            z4:= PBblas.PB_daxpy(1.0, z4_tmp, b1m);
      //a4 = sigmoid(z4)
            a4 := PBblas.Apply2Elements(b1map, z4, sigmoid);
            RETURN a4;
      END;//END FF4
```

```
   //Positive Product returns vhtrans
      PosProd (DATASET(Layout_Part) a2, DATASET(Layout_Part) ddist1) :=
FUNCTION
      //calculate vhtrans = X*a2trans
            vhtrans := PBblas.PB_dgemm(FALSE, TRUE, 1.0,dmap, ddist1,
a2map, a2, w2map);
            return vhtrans;
      END;//END DELTA3

      //Negative Product returns vdash_hdashtrans
      NegProd(DATASET(Layout_Part) a3, DATASET(Layout_Part) a4) :=
FUNCTION
      //calculate vdash_hdashtrans = a3*a4trans
            vdash_hdashtrans := PBblas.PB_dgemm(FALSE, TRUE, 1.0,a3map,
a3, a4map, a4, w2map);
            return vdash_hdashtrans;
      END;//END DELTA2

      //Gradient for weights
      WeightGrad1 (DATASET(Layout_Part) vhtrans, DATASET(Layout_Part)
vdash_hdashtrans, DATASET(Layout_Part) w2, DATASET(Layout_Part) deltaw)
:= FUNCTION
            w1_g := PBblas.PB_daxpy(-1.0, vdash_hdashtrans, vhtrans);
            term1 := PBblas.Apply2Elements(w2map, deltaw, applybeta); //
momentum*deltaW
            term2 := PBblas.Apply2Elements(w2map, w2,
applygamma);//eta*weight_cost*W
            term3 := PBblas.PB_daxpy(ALPHA/m, w1_g, term1);//eta*(pos-
neg) + momentum*deltaW
            term4 := PBblas.PB_daxpy(1.0, term3,
term2);//eta*weight_cost*W + eta*(pos-neg) + momentum*deltaW
            RETURN term4;
      END;//END WeightGrad1
      //Gradient for Bias1
      BiasGrad1 (DATASET(Layout_Part) a2, DATASET(Layout_Part) a4,
DATASET(Layout_Part) deltaupw) := FUNCTION
            term1 := PBblas.Apply2Elements(b1vecmap, deltaupw,
applybeta);//momentum*deltaBias_upW
            term2 := PBblas.PB_daxpy(-1.0, a4, a2);//hid1 - hid2
            term3 := PBblas.PB_dgemm(FALSE, FALSE, ALPHA*m_1, a2map,
term2, Ones_VecMap, Ones_Vecdist, b1vecmap);
            term4 := PBblas.PB_daxpy(1.0, term3, term1);
            RETURN term4;
      END;//END BiasGrad1

      //Gradient for Bias2
      BiasGrad2 (DATASET(Layout_Part) a3, DATASET(Layout_Part) ddist1,
DATASET(Layout_Part) deltadw) := FUNCTION
            term1 := PBblas.Apply2Elements(b2vecmap, deltadw,
applybeta);//momentum*deltaDownW
            term2 := PBblas.PB_daxpy(-1.0, a3, ddist1);//vis1 - vis2
            term3 := PBblas.PB_dgemm(FALSE, FALSE, ALPHA*m_1, a3map,
term2, Ones_VecMap, Ones_Vecdist, b2vecmap);//sum(vis1,1)-sum(vis2,1)
            term4 := PBblas.PB_daxpy(1.0, term1, term3);

            RETURN term4;
    END;//END BiasGrad2

      //Update Weights
      GradDesUpdate (DATASET(Layout_Part) tobeUpdated,
DATASET(Layout_Part) GradDesTerm):= FUNCTION
      //calculate as weight_updated := old_weight + ALPHA*diff_term
            tmp_updated := PBblas.PB_daxpy(ALPHA, GradDesTerm,
tobeUpdated);
            RETURN tmp_updated;
      END;//END GradDesUpdate

      //Update Bias
      BiasGradDesUpdate (DATASET(Layout_Part) tobeUpdated,
DATASET(Layout_Part) GradDesTerm):= FUNCTION
      //calculate as bias_updated := old_bias + diff_term
            tmp_updated := PBblas.PB_daxpy(1.0, GradDesTerm,
tobeUpdated);
            RETURN tmp_updated;
      END;//END BiasGradDesUpdate

//Gives the model with updated weights and bias matrices
      GradDesLoop (DATASET(Layout_Part) w1in, DATASET(Layout_Part) w2in,
DATASET(Layout_Part) bvec1in, DATASET(Layout_Part) bvec2in,
DATASET(Layout_Part) deltawin, DATASET(Layout_Part) deltaupwin,
DATASET(Layout_Part) deltadwin):= FUNCTION
            w1inno := PBblas.MU.TO(w1in, 1); // just a new field self.no
which is 1 for weight matrix 1
            w2inno := PBblas.MU.TO(w2in, 2); // just a new field self.no
which is 2 for weight matrix 2
            bvec1inno := PBblas.MU.TO(bvec1in, 3); // just a new field
self.no which is 3 for bias matrix 1
            bvec2inno := PBblas.MU.TO(bvec2in, 4);// just a new field
self.no which is 4 for bias matrix 2
            deltawinno := PBblas.MU.TO(deltawin, 15);// just a new field
self.no which is 15 for difference in weights
            deltaupwinno := PBblas.MU.TO(deltaupwin, 16);// just a new
field self.no which is 16 for difference in upward bias
            deltadwinno := PBblas.MU.TO(deltadwin, 17);// just a new
field self.no which is 17 for difference in downward bias
            prm := w1inno + w2inno + bvec1inno + bvec2inno + deltawinno +
deltaupwinno + deltadwinno;

      //Splits the input samples into batches for processing and also
repeats the loop for maxiter
            Split_Step(DATASET(PBblas.Types.MUElement) Inputprm2,
unsigned4 c) := FUNCTION
```

```
                  w1m1 := PBblas.MU.FROM (Inputprm2, 1); // weight
matrix1
                  w2m1 := PBblas.MU.FROM (Inputprm2, 2); //weight matrix2
                  b1v1 := PBblas.MU.FROM (Inputprm2, 3); //bias matrix1
                  b2v1 := PBblas.MU.FROM (Inputprm2, 4); //bias matrix2
                  deltawm1 := PBblas.MU.FROM (Inputprm2, 15);
                  deltaupwm1 := PBblas.MU.FROM (Inputprm2, 16);
                  deltadwm1 := PBblas.MU.FROM (Inputprm2, 17);
            //batch processing
                  innerloop := max(d,d.y)/batch_size;
            //maxiter processing
                  outerloop := if(c%innerloop=0,innerloop,c%innerloop);

            //Split the input samples
                  ddist1 := IF(c<=innerloop, SplitInput(X(id>=(c-
1)*batch_size+1 AND id<c*batch_size+1)),SplitInput(X(id>= (outerloop-
1)*batch_size+1 AND id<outerloop*batch_size+1))) ;

                  a2 := FF2 (w1m1, b1v1, ddist1);
                  a3 := BB3 (w2m1, b2v1, a2);
                  a4 := FF4 (w1m1, b1v1, a3);
                  vhtrans := PosProd (a2, ddist1);
                  vdash_hdashtrans := NegProd (a3, a4);
                  wg1 := WeightGrad1 (vhtrans, vdash_hdashtrans, w2m1,
deltawm1);
                  bg2 := BiasGrad2 (a3, ddist1, deltadwm1);
                  bg1 := BiasGrad1 (a2, a4, deltaupwm1);

                  w2u := GradDesUpdate (w2m1, wg1);
                  b1u := BiasGradDesUpdate (b1v1, bg1);
                  b2u := BiasGradDesUpdate (b2v1, bg2);

            //Convert the block weight matrix to original format for
obtaining transpose matrix and convert back again to block format
                  w2no_mat := DMat.Converted.FromPart2Elm (w2u);
                  w2no_mat_no := Mat.MU.TO(w2no_mat,1);
                  w1no_mat_no := Mat.Trans(w2no_mat_no);
                  w1uno_tmp :=
DMAT.Converted.FromElement(w1no_mat_no,w1map);

                  w2uno := PBblas.MU.TO (w2u, 2);
                  w1uno := PBblas.MU.TO(w1uno_tmp,1);
                  b1uno := PBblas.MU.TO (b1u, 3);
                  b2uno := PBblas.MU.TO (b2u, 4);
                  deltaw := PBblas.MU.TO (wg1, 15);
                  deltaupw := PBblas.MU.TO (bg1, 16);
                  deltadw := PBblas.MU.TO (bg2, 17);
                  retsplit := w1uno + w2uno + b1uno + b2uno + deltaw +
deltaupw + deltadw;
                  RETURN retsplit;
            END;
            flsplitstep := LOOP(prm, COUNTER <=
MaxIter*max(d,d.y)/batch_size, Split_Step(ROWS(LEFT), COUNTER));
        RETURN flsplitstep;
      END;//END GradDesLoop
    RBMprm := GradDesLoop (w1dist, w2dist, b1vecdist, b2vecdist,
Zeroes_Vecdist, Zeroes1_Vecdist, Zeroes2_Vecdist);
      //RBMprm contains combination of two weight matrices, two bias
matrices and diff_delta_matrices which are all zeroes initially
    //numericfield format
      RBMprm1 := PBblas.MU.From (RBMprm,1);//extract weight matrix1
      RBMprm2 := PBblas.MU.From (RBMprm,2);//extract weight matrix2
      RBMprm3 := PBblas.MU.From (RBMprm,3);//extract bias matrix1
      RBMprm4 := PBblas.MU.From (RBMprm,4);//extract bias matrix2
      RBMprm15 := PBblas.MU.From (RBMprm,15);
      RBMprm16 := PBblas.MU.From (RBMprm,16);
      RBMprm17 := PBblas.MU.From (RBMprm,17);
      RBMprm1_mat := DMat.Converted.FromPart2Elm (RBMprm1);
      RBMprm2_mat := DMat.Converted.FromPart2Elm (RBMprm2);
      RBMprm3_mat := DMat.Converted.FromPart2Elm (RBMprm3);
      RBMprm4_mat := DMat.Converted.FromPart2Elm (RBMprm4);
      RBMprm15_mat := DMat.Converted.FromPart2Elm (RBMprm15);
      RBMprm16_mat := DMat.Converted.FromPart2Elm (RBMprm16);
      RBMprm17_mat := DMat.Converted.FromPart2Elm (RBMprm17);
      RBMprm1_mat_no := Mat.MU.TO(RBMprm1_mat,1);
      RBMprm2_mat_no := Mat.MU.TO(RBMprm2_mat,2);
      RBMprm3_mat_no := Mat.MU.TO(RBMprm3_mat,3);
      RBMprm4_mat_no := Mat.MU.TO(RBMprm4_mat,4);
      RBMprm15_mat_no := Mat.MU.TO(RBMprm15_mat,15);
      RBMprm16_mat_no := Mat.MU.TO(RBMprm16_mat,16);
      RBMprm17_mat_no := Mat.MU.TO(RBMprm17_mat,17);
      RBMprm_MUE := RBMprm1_mat_no + RBMprm2_mat_no + RBMprm3_mat_no +
RBMprm4_mat_no + RBMprm15_mat_no + RBMprm16_mat_no + RBMprm17_mat_no; //
all weight, bias, delta matrices combined again
    AppendID(RBMprm_MUE, id, RBMprm_MUE_id);
    ToField (RBMprm_MUE_id, RBMprm_MUE_out, id, 'x,y,value,no');
    EXPORT Mod := RBMprm_MUE_out;
END;//END RBM

EXPORT LearnC (DATASET(Types.NumericField)
Indep,DATASET(Mat.Types.MUElement) IntW, DATASET(Mat.Types.MUElement)
Intb, REAL8 BETA=0.5, REAL8 LAMBDA=0.0002, REAL8 ALPHA=0.1, UNSIGNED2
MaxIter=100, UNSIGNED2 batch_size=10) := RBM(Indep,IntW,Intb,BETA,
LAMBDA, ALPHA,  MaxIter, batch_size).mod;
EXPORT Model(DATASET(Types.NumericField) mod) := FUNCTION
      modelD_Map :=
      DATASET([{'id','ID'},{'x','1'},{'y','2'},{'value','3'},{'no','4'}],
{STRING orig_name; STRING assigned_name;});
      FromField(mod,Mat.Types.MUElement,dOut,modelD_Map);
      RETURN dOut;
      END;//END Model
```

```
EXPORT RBMOutput(DATASET(Types.NumericField)
Indep,DATASET(Types.NumericField) mod) :=FUNCTION
      //Take the same steps in the FeedForward functions to calculate the
output of the resstricted boltzmann machine
      X := Indep;
      Inputmod:= Model (mod);
      dt := Types.ToMatrix (X);
      dTmp := dt;
      d := Mat.Trans(dTmp); //in the entire of the calculations we work
with the d matrix that each sample is presented in one column
      m := MAX (d, d.y); //number of samples
      sizeRec := RECORD
            PBblas.Types.dimension_t m_rows;
            PBblas.Types.dimension_t m_cols;
            PBblas.Types.dimension_t f_b_rows;
            PBblas.Types.dimension_t f_b_cols;
      END;
      //Map for Matrix d.

      havemaxrow := maxrows > 0;
      havemaxcol := maxcols > 0;
      havemaxrowcol := havemaxrow and havemaxcol;
      dstats := Mat.Has(d).Stats;
      d_n := dstats.XMax;
      d_m := dstats.YMax;
      derivemap := IF(havemaxrowcol, PBblas.AutoBVMap(d_n,
d_m,prows,pcols,maxrows, maxcols),
                  IF(havemaxrow, PBblas.AutoBVMap(d_n,
d_m,prows,pcols,maxrows),
                        IF(havemaxcol, PBblas.AutoBVMap(d_n,
d_m,prows,pcols,,maxcols),
                              PBblas.AutoBVMap(d_n, d_m,prows,pcols))));
                  sizeTable :=
DATASET([{derivemap.matrix_rows,derivemap.matrix_cols,derivemap.part_rows
(1),derivemap.part_cols(1)}], sizeRec);
      //Create block matrix d
      dmap :=
PBblas.Matrix_Map(sizeTable[1].m_rows,sizeTable[1].m_cols,sizeTable[1].f_
b_rows,sizeTable[1].f_b_cols);
    ddist := DMAT.Converted.FromElement(d,dmap);
    //Create block matrices for weights
    w1_mat := Mat.MU.From(Inputmod,1); // Weight matrix 1
    w1_mat_x := Mat.Has(w1_mat).Stats.Xmax;
    w1_mat_y := Mat.Has(w1_mat).Stats.Ymax;
    w1map := PBblas.Matrix_Map(w1_mat_x, w1_mat_y, sizeTable[1].f_b_rows,
sizeTable[1].f_b_rows);
    w1dist := DMAT.Converted.FromElement(w1_mat,w1map);
    w2_mat := Mat.MU.From(Inputmod,2); // Weight matrix 2
    w2_mat_x := w1_mat_y;
    w2_mat_y := w1_mat_x;
    w2map := PBblas.Matrix_Map(w2_mat_x, w2_mat_y, sizeTable[1].f_b_rows,
sizeTable[1].f_b_rows);
```

```
    w2dist := DMAT.Converted.FromElement(w2_mat,w2map);
    //each bias vector is converted to block format
    b1vec := Mat.MU.From(Inputmod,3); // Bias matrix 1
    b1vec_x := Mat.Has(b1vec).Stats.Xmax;
    b1vecmap := PBblas.Matrix_Map(b1vec_x, 1, sizeTable[1].f_b_rows, 1);
    b1vecdist := DMAT.Converted.FromElement(b1vec,b1vecmap);
    b2vec := Mat.MU.From(Inputmod,4); // Bias matrix 2
    b2vec_x := Mat.Has(b2vec).Stats.Xmax;
    b2vecmap := PBblas.Matrix_Map(b2vec_x, 1, sizeTable[1].f_b_rows, 1);
    b2vecdist := DMAT.Converted.FromElement(b2vec,b2vecmap);
    //functions used
    PBblas.Types.value_t sigmoid(PBblas.Types.value_t v, 
PBblas.Types.dimension_t r, PBblas.Types.dimension_t c) := 1/(1+exp(-
1*v));
    //maps used
    b1map := PBblas.Matrix_Map(b1vec_x, m, sizeTable[1].f_b_rows, 
sizeTable[1].f_b_cols);
    b2map := PBblas.Matrix_Map(b2vec_x, m, sizeTable[1].f_b_rows, 
sizeTable[1].f_b_cols);
    a2map := b1map;
    a3map := b2map;

    //onevec for calculating yhat
    Ones_VecMap := PBblas.Matrix_Map(m, 1, sizeTable[1].f_b_cols, 1);
    //New Vector Generator
    Layout_Cell gen(UNSIGNED4 c, UNSIGNED4 NumRows) := TRANSFORM
      SELF.x := ((c-1) % NumRows) + 1;
      SELF.y := ((c-1) DIV NumRows) + 1;
      SELF.v := 1;
    END;
    //Create Ones Vector for the calculations in the step fucntion
    Ones_Vec := DATASET(m, gen(COUNTER, m),DISTRIBUTED);
    Ones_Vecdist := DMAT.Converted.FromCells(Ones_VecMap, Ones_Vec);
    //b1m = repmat(b1v,1,m)
    b1m := PBblas.PB_dgemm(FALSE, TRUE, 1.0,b1vecmap, b1vecdist, 
Ones_VecMap, Ones_Vecdist, b1map);
    //z2 = w1*X+b1;
    z2 := PBblas.PB_dgemm(FALSE, FALSE, 1.0,w1map, w1dist, dmap, ddist, 
b1map, b1m, 1.0);
    //a2 = sigmoid (z2);
    a2 := PBblas.Apply2Elements(b1map, z2, sigmoid);
    a2_mat := DMat.Converted.FromPart2Elm(a2);

    NumericField tr (Mat.Types.Element le) := TRANSFORM
      SELF.id := le.y;
      SELF.number := le.x;
      SELF := le;
    END;
    RETURN PROJECT (a2_mat, tr(LEFT));
  END;//END SAOutput
```

```
EXPORT ExtractWeights (DATASET(Types.NumericField) mod) := FUNCTION
    SAmod := Model (mod);
    RETURN SAmod (no<3);
  END;//END ExtractWeights
  EXPORT ExtractBias (DATASET(Types.NumericField) mod) := FUNCTION
    SAmod := Model (mod);
    B := SAmod (no>2 AND no<5);
    Mat.Types.MUElement Sno (Mat.Types.MUElement l) := TRANSFORM
      SELF.no := l.no-2;
      SELF := l;
    END;
    RETURN PROJECT (B,Sno(LEFT));
  END;//END ExtractBias
  EXPORT ExtractW1 (DATASET(Types.NumericField) mod) := FUNCTION
    w1mod := mod (number = 4 and value = 1);
    Myid := RECORD
      w1mod.id;
    END;
    w1modid := TABLE(w1mod,Myid);
    w1r := JOIN (mod,w1modid,LEFT.id=RIGHT.id,TRANSFORM(LEFT) );
    RETURN w1r;
  END;
  EXPORT ExtractW2 (DATASET(Types.NumericField) mod) := FUNCTION
    w2mod := mod (number = 4 and value = 2);
    Myid := RECORD
      w2mod.id;
    END;
    w2modid := TABLE(w2mod,Myid);
    w2r := JOIN (mod,w2modid,LEFT.id=RIGHT.id,TRANSFORM(LEFT) );
    RETURN w2r;
  END;
  EXPORT Extractb1 (DATASET(Types.NumericField) mod) := FUNCTION
    b1mod := mod (number = 4 and value = 3);
    Myid := RECORD
      b1mod.id;
    END;
    b1modid := TABLE(b1mod,Myid);
    b1r := JOIN (mod,b1modid,LEFT.id=RIGHT.id,TRANSFORM(LEFT) );
    RETURN b1r;
  END;
  EXPORT Extractb2 (DATASET(Types.NumericField) mod) := FUNCTION
    b2mod := mod (number = 4 and value = 4);
    Myid := RECORD
      b2mod.id;
    END;
    b2modid := TABLE(b2mod,Myid);
    b2r := JOIN (mod,b2modid,LEFT.id=RIGHT.id,TRANSFORM(LEFT) );
    RETURN b2r;
  END;
```

```
EXPORT ExtractFinalOut (DATASET(Types.NumericField) mod) := FUNCTION
    finalmod := mod (number = 4 and value = 0);
    Myid := RECORD
      finalmod.id;
    END;
    finalmodid := TABLE(finalmod,Myid);
    finalr := JOIN (mod,finalmodid,LEFT.id=RIGHT.id,TRANSFORM(LEFT) );
    RETURN finalr;
  END;

END;//END RBM_Build

//this function stack ups NumRBMs restricted boltzmann machines to make a
Deep Belief Network
//In this module we receive unsupervised data and pass it through NumRBMs
layers of restricted boltzmann machines to initialize the weights in this
network with a Greedy Layer-Wise manner
//data is passed to the first RBM ( restricted boltzmann machine) and it
is trained, i.e. the weights are learnt, when it is trained the output of
it is passed to the second RBM as input, the second RBM is trained with
//this data, then the output of this RBM is passed as the input to the
third RBM, this continues until NumRBMs of RBMs are trained. At the end,
the whole network weights are initialized
//with this method
//NumRBMs : Number of RBMs in the Deep Belief Network, basically it means
number of restricted boltzmann machines that need to stack up to make the
deep belief network (the number of layers in the final Deep Learning
models is
//NumRBMs+1 because we have the input layer as well
//numHiddenNodes : number of hidden nodes in each restricted boltzmann
machine

EXPORT StackedRBM (UNSIGNED4 NumRBMs, DATASET(Types.DiscreteField)
numHiddenNodes, REAL8 BETA=0.5, REAL8 LAMBDA=0.0002, REAL8 ALPHA=0.1,
UNSIGNED2 MaxIter=100,
  UNSIGNED4 prows=0, UNSIGNED4 pcols=0, UNSIGNED4 Maxrows=0, UNSIGNED4
Maxcols=0, UNSIGNED2 batch_size=10) := MODULE
  NL := NumRBMs+1;//number of layers in the final Deep Learning algorithm
is 1 (input layer) + Number of restricted_boltzmann_machines

  DBN (DATASET(Types.NumericField) X) := MODULE
      //TRANFFORM used
      Mat.Types.MUElement Addno (Mat.Types.MUElement l, UNSIGNED v) :=
TRANSFORM
        SELF.no := l.no+v;
        SELF := l;
      END;

    //number of features in the input independent data
    NumFeatures := MAX (X,number);
```

```
     //Define the first Restricted Boltzmann Machine Module
    hd1 := numHiddenNodes(id=(1))[1].value;//number of hidden nodes in
the first RBM
    IntW1 := DBN_IntWeights(NumFeatures,hd1);//initialize weights
    Intb1 := DBN_IntBias(NumFeatures,hd1);//initialize bias
    RBM1 := RBM_Build (prows, pcols, Maxrows, Maxcols);//RBM module for
the first RBM
    //train the first Restricted Boltzmann Machine
    LearntModel1 := RBM1.LearnC(X,IntW1, Intb1, BETA, LAMBDA, ALPHA,
MaxIter, batch_size);//learnt model in NumericFiled format
    Bias1 := RBM1.ExtractBias (LearntModel1);
    Weight1 := RBM1.ExtractWeights (LearntModel1);
    RBMmodel1 := Weight1 (no=1) + PROJECT (Bias1 (no=1),Addno(LEFT,NL));
// Only weight and bias related to the first layer and hidden layer are
considered for each RBM to stack them up
    //produce the output of the first learnt Restricted Boltzmann Machine
    Output1 := RBM1.RBMOutput (X, LearntModel1);
    MatrixOutput1 := ML.Types.ToMatrix (Output1);
    MatrixOutput1No := Mat.MU.To(MatrixOutput1, 0);
   DBN_Step(DATASET(Mat.Types.MUElement) MM, INTEGER coun) := FUNCTION
      L := coun + 1;
      //output of the previous RBM which is going to be the input of the
next RBM
      lastOutput := Mat.MU.From(MM, 0);
      lastOutputF := ML.Types.FromMatrix(lastOutput);
      //Define the Lth Restricted_Boltzmann_Machine
      NFL := numHiddenNodes(id=(L-1))[1].value; //number of hidden layers
of the last RBM represents the number of input features for the next RBM
      hdL := numHiddenNodes(id=(L))[1].value;
      IntWL := DBN_IntWeights(NFL,hdL);//initialize weights
      IntbL := DBN_IntBias(NFL,hdL);//initialize bias
      RBML := RBM_Build (prows, pcols, Maxrows, Maxcols);//RBM module for
the Lth RBM
      //Train the Lth Restricted_Boltzmann_Machine (output of the last
RBM is fed as the input to the next RBM)
      LearntModelL := RBML.LearnC(lastOutputF,IntWL, IntbL, BETA, LAMBDA,
ALPHA, MaxIter, batch_size);
      BiasL := RBML.ExtractBias (LearntModelL);
      WeightL := RBML.ExtractWeights (LearntModelL);
      RBMmodelL := PROJECT (WeightL (no=1),Addno(LEFT,coun)) + PROJECT
(BiasL (no=1),Addno(LEFT,coun+NL));
      //produce the output of the Lth learnt Sparse Autoencoder
      OutputL := RBML.RBMOutput (lastOutputF, LearntModelL);
      MatrixOutputL := ML.Types.ToMatrix (OutputL);
      MatrixOutputLNo := Mat.MU.To(MatrixOutputL, 0);
      RETURN RBMmodelL + MatrixOutputLNo + MM (no > 0);
    END;//END DBN_Step
    EXPORT DBN_prm := LOOP(RBMmodel1 + MatrixOutput1No, COUNTER <=
NumRBMs-1, DBN_Step(ROWS(LEFT),COUNTER));//DBN_prm is in
Mat.Types.MUElement format convert it to NumericFieldFormat
    AppendID(DBN_prm, id, DBN_prm_id);
    ToField (DBN_prm_id, mm, id, 'x,y,value,no');//convert the learnt
model to numeric field before returning it
    EXPORT Mod := mm;
END;//END DBN
```

```
  //LearnC returns the learnt model from Stacking up of Restricted
Boltzmann Machines when some unsupervised data (Indep) are fed to it
  //the learnt model contains one weight and one bias matrix
correspondance to each Restricted Boltzmann Machine
  //the weight and bias matrix that correspond to each RBM are actually
the weight between first and hidden layer and the bias that goes to the
hideen layer
  //the output of the Restricted Boltzmann Machine (extracted feature)
has no =0

  EXPORT LearnC (DATASET(Types.NumericField) Indep) := DBN(Indep).Mod;
  /*
  //Model converts the learnt model from Numeric field format to the
Mat.Types.MUElement format
  //in the built model the no={1,2,..,NL-1} are the weight indexes
  //no={NL+1,NL+2,..,NL+NL-1} are bias indexes that go to the second,
third, ..,NL)'s layer respectively
  //no={1,NL+1}: weight and bias belong to the first SA
  //no={2,NL+2}: weight and bias belong to the second SA
  //no={NL-1,NL+NL-1}: weight and bias belong to the second NL-1th SA
  */
  EXPORT Model(DATASET(Types.NumericField) mod) := FUNCTION
    modelD_Map :=
      DATASET([{'id','ID'},{'x','1'},{'y','2'},{'value','3'},{'no','4'}],
{STRING orig_name; STRING assigned_name;});
    FromField(mod,Mat.Types.MUElement,dOut,modelD_Map);
    RETURN dOut;
  END;//END Model

      EXPORT StackedRBMOutput(DATASET(Types.NumericField) mod) :=
FUNCTION
                  RBML := RBM_Build (0, 0, 0, 0);
                  out := RBML.ExtractFinalOut(mod);
                  modelD_Map :=
      DATASET([{'id','ID'},{'x','1'},{'y','2'},{'value','3'},{'no','4'}],
{STRING orig_name; STRING assigned_name;});
                  FromField(out,Mat.Types.MUElement,dOut,modelD_Map);
                  RETURN dOut;
      END;a 8&xaertyuiop[

END;//StackedRBM

END;//END DBNN
```

References

1. Xue-wen Chen; Xiaotong Lin. 2014. "Big Data Deep Learning: Challenges and Perspectives," Access, IEEE, vol.2, no., pp.514, 525,doi: 10.1109/ACCESS.2014.2325029

2. Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. Journal of Big Data, 2(1), 1-21.

3. Van Der Maaten, L., Postma, E., & Van den Herik, J. (2009). Dimension reduction: a comparative. J Mach Learn Res, 10, 66-71.

4. In Noulas , A. K., & Krse, B. J. A. (2008, October). Deep Belief Networks for dimension reduction. In Belgian-Dutch Conference on Artificial Intelligence, Netherland.

5. Inselberg, A., & Dimsdale, B. (1987). Parallel coordinates for visualizing multi-dimensional geometry. In Computer Graphics 1987 (pp. 25-44). Springer Japan.

6. Keim, D. A., & Kriegel, H. P. (1994). VisDB: Database exploration using multidimensional visualization. Computer Graphics and Applications, IEEE, 14(5), 40-49.

7. Hoffman, P., Grinstein, G., & Pinkney, D. (1999, November). Dimensional anchors: a graphic primitive for multidimensional multivariate information visualizations. In Proceedings of the 1999 workshop on new paradigms in information visualization and manipulation in conjunction with the eighth ACM internation conference on Information and knowledge management (pp. 9-16). ACM.

8. Grinstein, G., Trutschl, M., & Cvek, U. (2001, August). High-dimensional visualizations. In Proceedings of the Visual Data Mining Workshop, KDD.

9. Agrawal, R., Kadadi, A., Dai, X., & Andres, F. (2015, October). Challenges and opportunities with big data visualization. In Proceedings of the 7th International Conference on Management of computational and collective intElligence in Digital EcoSystems (pp. 169-173). ACM.

10. Gering, D. (2002). Linear and nonlinear data dimension reduction. fullfillment of the Area Exam doctoral requirements.

11. Teli, M. N. (2007). Dimension reduction using neural networks. In Intelligent Engineering Systems Through Artificial Neural Networks, Volume 17. ASME Press.

12. Van Der Maaten, L., Postma, E., & Van den Herik, J. (2009). Dimension reduction: a comparative. J Mach Learn Res, 10, 66-71.

13. Tsai, F. S. (2011). Dimension reduction techniques for blog visualization. Expert Systems With Applications, 382766-2773. doi:10.1016/j.eswa.2010.08.067

14. Claveria, O., & Poluzzi, A. (2016). Research paper: Positioning and clustering of the world's top tourist destinations by means of dimension reduction techniques for categorical data. Journal Of Destination Marketing & Management, doi:10.1016/j.jdmm.2016.01.008

15. HPCC Systems, 2016: http://hpccsystems.com, accessed in April 2016

16. Deep Learning Tutorial Release 0.1, LISA Lab, University of Montreal, March 09, 2015.

17. Neukart F, Moraru S. A Machine Learning Approach for Abstraction based on the Idea of Deep Belief Artificial Neural Networks. Procedia Engineering [serial online]. January 1, 2014;69(24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013):1499-1508. Available from: ScienceDirect, Ipswich, MA. Accessed May 9, 2015.

18. Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., & Lew, M. S. (2016). Deep learning for visual understanding: A review. Neurocomputing, 187(Recent Developments on Deep Big Vision), 27-48. doi:10.1016/j.neucom.2015.09.116

19. Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol.2010.Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion, Journal of Machine Learning Research 11 (2010) 3371-3408.

20. Stanford Tutorial, 2015: http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders, accessed in July 2015.

21. Zhang, K., Liu, J., Chai, Y., & Qian, K. (2015). An optimized dimension reduction model for high-dimensional data based on Restricted Boltzmann Machines. Proceedings Of The 2015 27Th Chinese Control And Decision Conference, CCDC 2015, (Proceedings of the 2015 27th Chinese Control and Decision Conference, CCDC 2015), 2939-2944. doi:10.1109/CCDC.2015.7162428

22. Liu, Y., Zhou, S., & Chen, Q. (2011). Discriminative deep belief networks for visual data classification. Pattern Recognition, 44(Semi-Supervised Learning for Visual Content Analysis and Understanding), 2287-2296. doi:10.1016/j.patcog.2010.12.012

23. Tsai, F. S. (2012). A visualization metric for dimension reduction. Expert Systems With Applications, 391747-1752. doi:10.1016/j.eswa.2011.08.080

24. Kalina, J., & Rensová, D. (2015). HOW TO REDUCE DIMENSION OF DATA: ROBUSTNESS POINT OF VIEW. Serbian Journal Of Management, 10(1), 131-140. doi:10.5937/sjm10-6531

25. Wang, Y., Yao, H., Zhao, S., & Zheng, Y. (2015, August). Dimension reduction strategy based on auto-encoder. In Proceedings of the 7th International Conference on Internet Multimedia Computing and Service (p. 63). ACM.

26. Venna, J., Peltonen, J., Nybo, K., Aidos, H., & Kaski, S. (2010). Information Retrieval Perspective to Nonlinear Dimension Reduction for Data Visualization. Journal Of Machine Learning Research, 11(2), 451-490.

27. Najim, S. A., & Lim, I. S. (2014). Trustworthy dimension reduction for visualization different data sets. Information Sciences, 278206-220. doi:10.1016/j.ins.2014.03.048

28. Dzwinel, W., & Wcisło, R. (2015). Very Fast Interactive Visualization of Large Sets of High-dimensional Data. Procedia Computer Science, 51(International Conference On Computational Science, ICCS 2015), 572-581. doi:10.1016/j.procs.2015.05.325

29. Wang, Q., & Li, J. (2009). Combining local and global information for nonlinear dimension reduction. Neurocomputing, 72(10), 2235-2241.

30. Bikku, T., Rao, N. S., & Akepogu, A. R. (2016). Hadoop based Feature Selection and Decision Making Models on Big Data. Indian Journal of Science and Technology, 9(10).

31. Sandy Ryza, Uri Laserson, Sean Owen, Josh Wills . (2015). Advance Analytics With Spark. Publisher : Beijing; Sebastopol, CA : O'Reilly

32. Mohamad, I. (2013). Standardization and its effects on k-means clustering algorithm. Research Journal of Applied Sciences, Engineering and Technology, 6(17), 3299-3303.

33. Visalakshi, N. K., & Thangavel, K. (2009). Impact of normalization in distributed k-means clustering. International Journal of Soft Computing, 4(4), 168-172.

34. Al Shalabi, L., Shaaban, Z., & Kasasbeh, B. (2006). Data mining: A preprocessing engine. Journal of Computer Science, 2(9), 735-739.

35. Burkardt, J. (2009). K-Means Clustering. Virginia Tech, Advanced Research Computing, Interdisciplinary Center for Applied Mathematics.

36. Cui, X., Zhu, P., Yang, X., Li, K., & Ji, C. (2014). Optimized big data K-means clustering using MapReduce. Journal Of Supercomputing, 70(3), 1249. doi:10.1007/s11227-014-1225-7

37. Sigillito, V. G., Wing, S. P., Hutton, L. V., \& Baker, K. B. (1989). Classification of radar returns from the ionosphere using neural networks. Johns Hopkins APL Technical Digest, 10, 262-266.

38. Wolberg, W.H., & Mangasarian, O.L. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. In Proceedings of the National Academy of Sciences, 87, 9193--9196.

39. Zhang, J. (1992). Selecting typical instances in instance-based learning. In Proceedings of the Ninth International Machine Learning Conference (pp. 470--479). Aberdeen, Scotland: Morgan Kaufmann.

40. S. Aeberhard, D. Coomans and O. de Vel, Comparison of Classifiers in High Dimensional Settings, Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland.

41. S. Aeberhard, D. Coomans and O. de Vel, "THE CLASSIFICATION PERFORMANCE OF RDA" Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland.

42. Michie,D. (1988). The Fifth Generation's Unbridged Gap. In Rolf Herken (Ed.) The Universal Turing Machine: A Half-Century Survey, 466-489, Oxford University Press.

43. Paul Horton & Kenta Nakai. "A Probablistic Classification System for Predicting the Cellular Localization Sites of Proteins".Intelligent Systems in Molecular Biology, 109-115. St. Louis, USA 1996.

44. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, accessed in May 2016

45. Anthony M.Middleton, Ph.D., 2011-05, HPCC Systems: Introduction to HPCC

46. HPCC cluster, 2016 https://216.19.105.7:18010 accessed in April 2016

47. Geoffrey Hinton, Department of Computer Science, University of Toronto, Version 1, 2010, A Practical Guide to Training Restricted Boltzmann Machines