DISC: Dimensionality Reduction Using Sample Clustering

Harshit Mahapatra
Application Number: ENGS5079,
Guide: Dr. Nikhil R. Pal

Indian Statistical Institute, Kolkata

Abstract

Advances in data collection and storage capabilities during the past decades have led to an information overload in most sciences. Researchers working in diverse domains face larger and larger observations and simulations on a daily basis. Such datasets present new challenges in data analysis. Traditional statistical methods break down because of the increase in the number of variables associated with each observation. In this project an algorithm named DISC i.e. Dimensionality reductIon using Sample Clustering is proposed which uses cluster analysis for reducing the dimensionality of such datasets. The proposed algorithm was tested on numerous datasets and its performance was noted.

Keywords: Dimensionality Reduction, Clustering, High Dimensional datasets

Contents

1. Introduction

High-dimensional datasets present many mathematical challenges as well as some opportunities, and are bound to give rise to new theoretical developments. One of the problems with high-dimensional datasets is that, in many cases, not all the measured variables are important for understanding the underlying phenomena of interest.

One common term associated with analysis of high dimensional datasets is called *"the curse of dimensionality."* It can be defined as various phenomena that arise when analyzing and organizing data in high-dimensional spaces (often with hundreds or thousands of dimensions) that do not occur in low-dimensional settings. Here, when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of

data needed to support the result often grows exponentially with the dimensionality. Also, organizing and searching data often relies on detecting areas where objects form groups with similar properties; in high dimensional data, however, all objects appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient.

While certain computationally expensive novel methods can construct predictive models with high accuracy from high-dimensional data, it is still of interest in many applications due to the aforementioned reasons to reduce the dimension of the original data prior to any modeling of the data. In mathematical terms, the problem we investigate can be stated as follows: given the p dimensional random variable $x = (x_1, ..., x_p)^T$, find a lower dimensional representation of it, $s = (s_1, ..., s_k)^T$ with $k \leq p$, that captures the content in the original data, according to some criterion. The components of s are sometimes called the hidden components. Different fields use different names for the p multivariate vectors: the term variable is mostly used in statistics, while feature and attribute are alternatives commonly used in the computer science and machine learning literature.

2. Popular Dimensionality Reduction Techniques

In this section we will look at the existing popular methods that are used for dimensionality reduction in brief.

The following are the most popular state-of-the-art dimesionality reduction techniques currently available and accepted in data analytics landscape:

1. Missing Values Ratio
2. Low Variance Filter
3. High Correlation Filter
4. Random Forests
5. Principal Component Analysis
6. Backward Feature Elimination
7. Forward Feature Construction

2.1. Missing Values Ratio

Data columns with too many missing values are unlikely to carry much useful information. Thus data columns with number of missing values greater than a given threshold can be removed. The higher the threshold, the more aggressive the reduction.

2.2. Low Variance Filter

The variance of a random variable X is the expected value of the squared deviation from the mean of X. It is given by the formula $= E[X]$.Data columns with little changes in the data carry little information. Thus all data columns with variance lower than a given threshold are removed. However it should be done cautiously as variance is range dependent, therefore before applying this technique, normalization is needed.A random variable X (feature) is standardized by subtracting its expected value $E[X]$ and dividing the difference by its standard deviation $\sigma(X) = \sqrt{Var(X)}$. Thus the normalized columns are represented by the formula:

$$Z = \frac{X - E[x]}{\sigma(x)} \tag{1}$$

2.3. High Correlation Filter

Data columns with very similar trends are also likely to carry very similar information. In this case, only one of them will suffice to feed the machine learning model. Here we calculate the correlation coefficient between numerical columns and between nominal columns as the Pearsons Product Moment Coefficient and the Pearsons chi square value respectively.

Pearson's correlation coefficient is the covariance of the two variables divided by the product of their standard deviations.It is a measure of the linear correlation between two variables X and Y, giving a value between +1 and 1 inclusive, where 1 is total positive correlation, 0 is no correlation, and 1 is total negative correlation. It is widely used in the sciences as a measure of the degree of linear dependence between two variables. It is given by the formula

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{2}$$

Here $cov(X,Y)$ is the covariance, and σ is the standard deviation.

Pairs of columns with correlation coefficient higher than a threshold are reduced to only one. This method should be applied with caution as correlation is scale sensitive, therefore column normalization is required for a meaningful correlation comparison.

2.4. Random Forests

Decision Tree Ensembles, also referred to as random forests, are useful for feature selection in addition to being effective classifiers. One approach

to dimensionality reduction is to generate a large and carefully constructed set of trees against a target attribute and then use each attributes usage statistics to find the most informative subset of features. Specifically, we can generate a large set of very shallow trees, with each tree being trained on a small fraction of the total number of attributes. If an attribute is often selected as best split, it is most likely an informative feature to retain. A score calculated on the attribute usage statistics in the random forest tells us relative to the other attributes which are the most predictive attributes.

$$\begin{aligned} Score = \{\#splits(lev.0)/\#candidates(lev.0)\}+ \\ \{\#splits(lev.1)/\#candidates(lev.1)\} \end{aligned} \tag{3}$$

This score tells us relative to the other attributes which are the most predictive. Only the input features scoring higher than a given threshold are retained.This techniques produces a strong reduction rate, while minimally affecting the original accuracy.

2.5. Principal Component Analysis

Principal Component Analysis (PCA) is a statistical procedure that orthogonally transforms the original n coordinates of a data set into a new set of n coordinates called principal components. As a result of the transformation, the first principal component has the largest possible variance; each succeeding component has the highest possible variance under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding components. Keeping only the first m ¡ n components reduces the data dimensionality while retaining most of the data information, i.e. the variation in the data. Notice that the PCA transformation is sensitive to the relative scaling of the original variables. Data column ranges need to be normalized before applying PCA. Also the new coordinates (PCs) are not real system-produced variables anymore. Applying PCA to a data set loses its interpretability.

Consider a data matrix, X, with column-wise zero empirical mean (the sample mean of each column has been shifted to zero), where each of the n rows represents a different repetition of the experiment, and each of the p columns gives a particular kind of feature (say, the results from a particular sensor).

Mathematically, the transformation is defined by a set of p-dimensional vectors of weights or loadings $w_{(k)} = (w_1, ..w_p)$ of row vector $X_{(i)}$ of X to a

new vector of principal components scores $t_{(i)} = (t_1, ..., t_k)_{(i)}$ given by

$$t_{k(i)} = x_{(i)} \cdot w_{(k)} \tag{4}$$

in such a way that the individual variables of t considered over the data set successively inherit the maximum possible variance from x, with each loading vector w constrained to be a unit vector.

For dimensionality reduction, transformation $T = XW$ maps a data vector $x_{(i)}$ from an original space of p variables to a new space of p variables which are uncorrelated over the dataset. However, not all the principal components need to be kept. Keeping only the first L principal components, produced by using only the first L loading vectors, gives the truncated transformation

$$T_L = XW_L \tag{5}$$

where the matrix T_L now has n rows but only L columns. In other words, PCA learns a linear transformation $t = W^T x, x \in R^p, t \in R^L$,where the columns of $p \times L$ from matrix W form an orthogonal basis for the L features (the components of representation t) that are decorrelated.By construction, of all the transformed data matrices with only L columns, this score matrix maximises the variance in the original data that has been preserved, while minimising the total squared reconstruction error given by:

$$||TW^T - T_L W_L^T||_2^2 \quad or \quad ||X - X_L||_2^2 \tag{6}$$

The PCA algorithm for dimensionality reduction can be summarized as follows:

1. Compute covariance matrix $C_x, C_x = \frac{1}{n} XX^T$
2. We select the matrix P to be a matrix where each row p_i is an eigenvector of $\frac{1}{n} XX^T$
3. If A is a square matrix, a non-zero vector v is an eigenvector of A if there is a scalar such that $Av = \lambda v$
4. Reduction: there are m eigenvectors, we reduce from m dimensions to k dimensions by choosing k eigenvectors related with k largest eigenvalues λ .

The value of k is choosen on the basis of proportion of Variance (PoV) explained by the k eigenvectors i.e.

$$PoV = \frac{\lambda_1 + \lambda_2 + \cdots \lambda_k}{\lambda_1 + \lambda_2 + \cdots \lambda_k + \cdots + \lambda_m} \tag{7}$$

where λ_i are sorted in descending order. Typically k is choosen such that $PoV0.9$. It should be noted that despite its many advantages there are some situations in which PCA algorithms does not give satisfactory performance. For example PCA performs poorly when,

- The data is non-linear
- The data is not normally distributed.
- Variance in data is due to error.

Figure 1 shows the scatterplot of the top two principal components of the genetic dataset GLA-BRA-180 consisting of 180 samples and 49151 features with 4 classes. The scatterplot helps us in visualizing the structure of otherwise complex dataset.

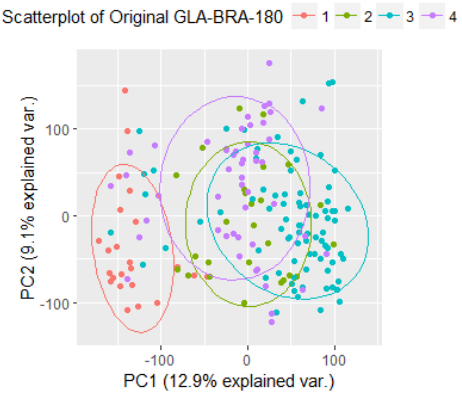

Figure 1: The scatterplot of first two principal components of genetic dataset GLA-BRA-180.

2.6. Backward Feature Elimination

In this technique, at a given iteration, the selected classification algorithm is trained on n input features. Then we remove one input feature at a time and train the same model on n-1 input features n times. The input feature whose removal has produced the smallest increase in the error rate

is removed, leaving us with n-1 input features. The classification is then repeated using n-2 features, and so on. Each iteration k produces a model trained on n-k features and an error rate e(k). Selecting the maximum tolerable error rate, we define the smallest number of features necessary to reach that classification performance with the selected machine learning algorithm.

2.7. Forward Feature Construction

This is the inverse process to the Backward Feature Elimination. We start with 1 feature only, progressively adding 1 feature at a time, i.e. the feature that produces the highest increase in performance. Both algorithms, Backward Feature Elimination and Forward Feature Construction, are quite time and computationally expensive. They are practically only applicable to a data set with an already relatively low number of input columns.

3. The K-Means Algorithm and Cluster Analysis

In this section we will breifly discuss about clustering and the K-means algorithm.

3.1. Clustering

Clustering or cluster analysis is a division of data into groups of similar objects. Each group, called cluster, consists of objects that are similar between themselves and dissimilar to objects of other groups. Representing data by fewer clusters necessarily loses certain fine details (akin to lossy data compression), but achieves simplification. It represents many data objects by few clusters, and hence, it models data by its clusters. Data modeling puts clustering in a historical perspective rooted in mathematics, statistics, and numerical analysis. From a machine learning perspective clusters correspond to hidden patterns, the search for clusters is unsupervised learning, and the resulting system represents a data concept. Therefore, clustering is unsupervised learning of a hidden data concept. Data mining deals with large databases that impose on clustering analysis additional severe computational requirements.

Formally we can define clustering as given a dataset X consisting of data points (or synonymously, objects, instances, cases, patterns, tuples, transactions) $x_i = (x_{i1}, ..., x_{id}) \in A$ in attribute space A, where $i = 1 : N$, and each component is a numerical or nominal categorical attribute (or synonymously, feature, variable, dimension, component, field).

The ultimate goal of clustering is to assign points to a finite system of k subsets, clusters. Usually subsets do not intersect (this assumption is sometimes violated), and their union is equal to a full dataset with possible exception of outliers.

$$X = C_1 \cup ... \cup C_k \cup C_{outliers}, C_{j1} \cap C_{j2} = \emptyset \tag{8}$$

3.2. The K-means Algorithm

The most popular and the simplest partitional algorithm is the K-means algorithm.. K-means has a rich and diverse history as it was independently discovered in different scientic elds by Steinhaus (1956), Lloyd (proposed in 1957, published in 1982), Ball and Hall (1965), and MacQueen (1967). Even though K-means was rst proposed over 50 years ago, it is still one of the most widely used algorithms for clustering. Ease of implementation, simplicity, efciency, and empirical success are the main reasons for its popularity.

Let $X = \{x_i\}i = 1, ..., n$ be the set of n d-dimensional points to be clustered into a set of K clusters, $C = \{c_k; k = 1, ..., K\}$. K-means algorithm nds a partition such that the squared error between the empirical mean of a cluster and the points in the cluster is minimized. Let l_k be the mean of cluster c_k. The squared error between l_k and the points in cluster c_k is dened as

$$J(c_k) = \sum_{x_i \in c_k} ||x_i - u_k||^2 \tag{9}$$

The goal of K-means is to minimize the sum of the squared error over all K clusters,

$$J(C) = \sum_{k=1}^{K} \sum_{x_i \in c_k} ||x_i - u_k||^2 \tag{10}$$

Minimizing this objective function is known to be an NP-hard prob- lem (even for K =2)(Drineas et al., 1999). Thus K-means, which is a greedy algorithm, can only converge to a local minimum, even though recent study has shown with a large probability K-means could converge to the global optimum when clusters are well sep- arated (Meila, 2006).The main steps of K-means algorithm are as follows (Jain and Dubes, 1988):

1. Select an initial partition with K clusters; repeat steps 2 and 3 until cluster membership stabilizes.

2. Generate a new partition by assigning each pattern to its closest cluster center.
3. Compute new cluster centers

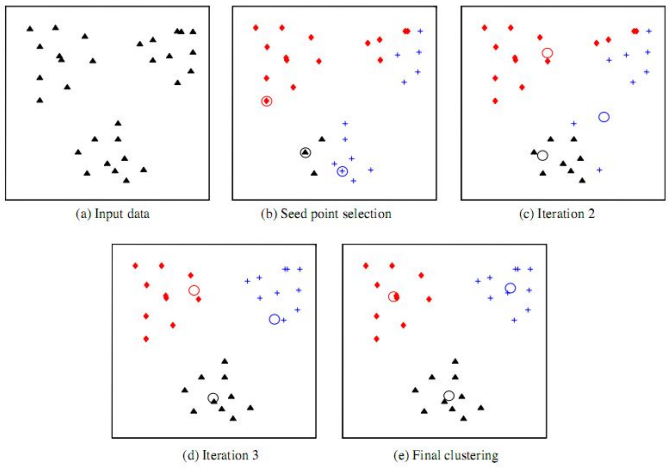

Figure 2: Output of k-means algorithm during various iterations.

The K-means algorithm requires three user-specied parameters: number of clusters K, cluster initialization, and distance metric. The most critical choice is K. While no perfect mathematical criterion exists, a number of heuristics are available for choosing K. Typically, K-means is run independently for different values of K and the partition that appears the most meaningful to the domain expert is selected. Different initializations can lead to different nal clustering because K-means only converges to local minima. One way to overcome the local minima is to run the K-means algorithm, for a given K, with multiple different initial partitions and choose the partition with the smallest squared error.

K-means is typically used with the Euclidean metric for computing the distance between points and cluster centers. As a result, K-means nds spherical or ball-shaped clusters in data. K-means with Mahalanobis distance metric has been used to detect hyper-ellipsoidal clusters, but this comes at the expense of higher computational cost.

4. Related Work

In this section we will discuss about Multi-label learning and the LIFT algorithm proposed by Min-Ling Zhang and Lei Wu.

4.1. Multi Label Learning

Multi-label learning aims to build classication models for objects assigned-with multiple class labels simultaneously.Multi-label objects widely exist in various real-world applications, such as text categorization where a news document could cover several topics including politics, economics, and reform,multimedia content annotation where one image could demonstrate several scenes including beach and building bioinformatics where one gene could have a number of functionalities including metabolism, protein synthesis, and transcription.

Formally, the problem can be defined as follows: If $X = R^d$ denote the d-dimensional input space and $Y = \{l_1, l_2, ..., l_q\}$ denote the label space with q class labels. Then, the task of multi-label learning is to derive a multi-label classication function $h : X \to 2^Y$ which assigns each instance $x \in X$ with a set of relevant labels $h(x) \subseteq Y$.

It is evident that traditional supervised learning can be regarded as a degenerated version of multi-label learning if each example is conned to have only one single label. However, the generality of multi-label learning inevitably makes the corresponding learning task much more difcult to solve. Actually, the key challenge of learning from multi-label data lies in the overwhelming size of output space, i.e. the number of label sets grows exponentially as the number of class labels increases. For example, for a label space with 20 class labels (q =20), the number of possible label sets would exceed one million (i.e. 220).

4.2. The LIFT Algorithm

LIFT stands for "multi-label learning with **L**abel spec**I**fic **F**ea**T**ures". It was proposed by Min-Ling Zhang and Lei Wu as an approach to multi-label learning.

Given a set of m multi-label training examples $D = \{(x_i, y_i)|1 \leq i \leq m\}$ where $x_i \in X$ is a d-dimensional feature vector and $y_i \in Y$ is the set of relevant labels associated with x_i. LIFT learns from D by taking two elementary steps, i.e. label-specic features construction and classication models induction.

In the *first* step, LIFT aims to construct features which could effectively capture the specic characteristics of each label, so as to provide appropriate distinguishing information to facilitate its discrimination process.the training instances with respect to each class label. Specifically, for one class label $l_k \in Y$, the set of positive training instances P_k as well as the set of negative training instances N_k are calculated as follows:

$$\begin{aligned} P_k &= \{x_i | (x_i, y_i) \in D, l_k \in y_i\} \\ N_k &= \{x_i | (x_i, y_i) \in D, l_k \notin y_i\} \end{aligned} \tag{11}$$

Now P_k is partitioned into m_k^+ disjoint clusters whose centers are denoted as $\{p_1^k, p_2^k, ..., p_{m_k^+}^k\}$. Similarly, N_k is partitioned into m_k^- disjoint clusters whose centers are denoted as $\{n_1^k, n_2^k, ..., n_{m_k^-}^k\}$. However generally we encounter the issue of class imbalance i.e. $|P_k| \ll |N_k|$. In order to resolve this LIFT sets an equivalent number of clusters m_k given by:

$$m_k = \lceil r \cdot min(|P_k|, |N_k|) \rceil \tag{12}$$

Here $r \in [0, 1]$ is the ratio parameter which controls the number of clusters retained.

Now, a mapping $\phi_k : X \to Z_k$ from the original d-dimensional input space X to the $2m_k$-dimensional label-specic feature space is created as follows:

$$\phi_k(x) = [d(x, p_1^k), ..., d(x, p_{m_k}^k), d(x, n_1^k), ..., d(x, n_{m_k}^k)] \tag{13}$$

Here,$d(.,.)$ returns the distance between two instances.

In the second step, a family of q classication models $\{g_1, g_2, ..., g_q\}$are induced with the generated label-specic features. Here, for each class label $l_k \in Y$, a new binary training set B_k with m examples is created from the original multi-label training set D and the mapping ϕ_k as follows:

$$\begin{aligned} B_k &= \{(\phi_k(x_i), Y_i(k)) | (x_i, Y_i) \in D\} \quad where \\ Y_i(k) &= +1 if l_k \in Y_i; \quad Otherwise Y_i(k) = -1 \end{aligned} \tag{14}$$

Based on B_k, any binary learner L can be applied to induce classication model $gk : Z_k \to R$ for l_k. Given an unseen example $u \in X$, its associated label set i predicted as:

$$Y = \{l_k | g_k(\phi_k(u)) > 0, 1 \leq k \leq q\} \tag{15}$$

The LIFT algorithm performs at par and in some cases, even better than the existing well established multi-label learning algorithms like Binary relevance,Calibrated label ranking,Ensemble of classifier chains,etc.

4.3. Disadvantages of LIFT

LIFT algorithm despite its many advantages has the folowing disadvantages:

1. **Class information is required:** For reducing a dataset using LIFT, we require the class information of each instance. This makes LIFT unsuitable for unsupervised dimensionality reduction.

2. **Space Requirement:** LIFT creates a new binary dataset of the size of the original for each class type. Thus a lot of space is required and this makes LIFT unsuitable for reducing large datasets.

3. **Use of euclidean distance in High Dimension:** Although LIFT can be used with different distance metric, the default metric used was euclidean. As a result in LIFT euclidean distance was used in high dimension spaces. This is not desirable due to the odd behaviour of euclidean distance in high dimension as suggested in the paper "On the Surprising Behavior of Distance Metrics in High Dimensional Space" by Aggarwal, Hinneburg and Keim.

5. Proposed Algorithm

In this section we will breifly discuss about the proposed algorithm.

5.1. The DISC Algorithm

Given a set of m data instances $X = \{x_i | 1 \leq i \leq m\}$ where x_i is a n-dimensional feature vector, DISC reduces the dataset X in two steps i.e. *sample clustering* and *feature construction*.

In the *first* step, DISC divides up the total dataset into a number of equal smaller datasets by feature. The number of features in each dataset represented by s is the first parameter of the algorithm. Mathematically the smaller datasets can be defined as follows:

$$
\begin{aligned}
&X = X_1 \cup X_2 \cup ... \cup X_{n_s} \quad where \\
&n_s = \left\lfloor \frac{n}{s} \right\rfloor if \quad \bmod \frac{n}{s} < \frac{s}{2} \\
&n_s = \left\lfloor \frac{n}{s} \right\rfloor + 1 \quad Otherwise
\end{aligned} \tag{16}
$$

X_i contains features in the range $[s * (i-1), s * i]$. It should be noted that the last subset can contain features in the range $[s * (\lfloor \frac{n}{s} \rfloor - 1), n]$ if mod $\frac{n}{s} < \frac{s}{2}$. Otherwise, the last subset will contain features in the range $[s * (\lfloor \frac{n}{s} \rfloor), n]$. This heuristic is adopted so that the dataset is divided as evenly as possible.

Now, K-means clustering is performed on each subset X_i to get c centers. c is the second parameter of DISC algorithm. Thus for each X_i we get c s-dimensional vectors given by:

$$
C_i = \{c_1^i, c_2^i, ..., c_c^i\}. \tag{17}
$$

In the *second* step, for each subset X_i we define a mapping $\phi : R^s \rightarrow R^c$ given by:

$$
\hat{X}_i = \phi(X_i) = \{d(X_i, c_1^i), d(X_i, c_2^i), ..., d(X_i, c_c^i)\} \tag{18}
$$

Here $d(.,.)$ returns euqlidean distance between each instance of X_i and the s-dimensional vectors $\{c_j^i | 1 \leq j \leq c\}$. $\hat{X}_i$ represents the i-th subset of reduced dataset. Finally we can get the reduced dataset as:

$$
\hat{X} = \hat{X}_1 \bowtie \hat{X}_2 \bowtie ... \bowtie \hat{X}_{n_s} \tag{19}
$$

Thus if the original dataset is of n dimensions then the reduced dataset is of p dimensions, where the value of p is:

$$
p = \frac{n}{n_s} * c \tag{20}
$$

where the value of n_s is given by equation 16.

5.2. DISC Pseudocode

The pseudocode of DISC algorithm is given below:

Algorithm 1: DISC Algorithm

$\hat{X} = DISC(X, s, c)$

Input :

X : A dataset s $\{x_i | 1 \leq i \leq m, x_i \in R^n\}$

s : An integer to determine the subsetsize $s \in I$.

c : An integer determining the number of centers into which the samples are clustered $c \in I$

Output:

$\hat{X}$: A dataset with reduced dimensions $\{\hat{x}_i \in R^p | p \leq n\}$

Process:

1 $\hat{X} = \emptyset$

2 Divide X into n_s subsets of size s according to Eq.(16).

3 **for** $i = 1$ *to* n_s **do**

4 Apply K-means on X_i with c centers to get C_i according to Eq(17).

5 Compute $\hat{X}_i$ according to Eq.(18).

6 $\hat{X} = \hat{X} \bowtie \hat{X}_i$

7 **end**

5.3. Advantages of DISC

We can clearly see that DISC overcomes each of the demerits of LIFT i.e.

1. **DISC does not require class information:** DISC algorithm is indifferent to the class of an observation and is thus suitable for unsupervised dimensionality reduction.
2. **DISC reduces the space required to store the data:** Unlike LIFT, DISC reduces the space required to store the data depending upon the passed parameters.
3. **DISC uses euclidean in reasonable lower dimensions:** In DISC the total feature set is divided into a number of subsets thus reducing the effective dimensionality of the data on which euclidean distance is to be calculated.
4. **Parallelization:** In DISC the clustering of each subset is independent of another thus subsets can be reduced in a parallel manner.

6. Experimentation

For checking the effectiveness of DISC experiments were conducted on both single label and multi label datasets.

6.1. Single Label Datasets

The following single-label datasets were used in the project:

Dataset	Samples	Features	Classes
GLA-BRA-180	180	49151	4
SMK-CAN-187	187	19993	2
Gisette	7000	5000	2

(Note:) In every experiment the dataset used was randomly split into training and test set in a 50-50 ratio.

Experiment 1:

1. First to check the effectiveness of DISC datasets GLA-BRA-180 and SMK-CAN-187 were both reduced to approximately $\frac{1}{10}$ of their original dimension using both euclidean and cosine similarity as distance metric for the mapping function in Eq(18).
2. Then K-nearest neighbour classifier was run on the original and the reduced dimension with varying parameters and its performance in terms of accuracy was noted.
3. The process was repeated 10 times and the average accuracies were calculated.

The results of the experiments are as follows:

	Original			Reduced(Euclid)			Reduced(Cosine)		
k	3	5	7	3	5	7	3	5	7
GLA-BRA-180	0.645	0.665	0.666	0.658	0.658	0.66	0.651	0.668	0.671
SMK-CAN-187	0.633	0.656	0.668	0.655	0.648	0.658	0.625	0.617	0.642

Observation: It was observed that performance of Knn classifier was similar on both original and reduced datasets. Furthermore it was observed that

the performance of classifier was similar on datasets reduced with euclidean distance to thst of its performance on datasets reduced wuth cosine similarity.

Note: From here on libsvm was used for classification.

Experiment 2:

1. To see the variation of DISC with its parameter r, it was run on Gisette dataset with varying values of r and the accuracies were noted. The value of parameter s was fixed as 50 during the procedure.
2. The process was repeated 10 times and the average accuracies were plotted.

The following figure was obtained:

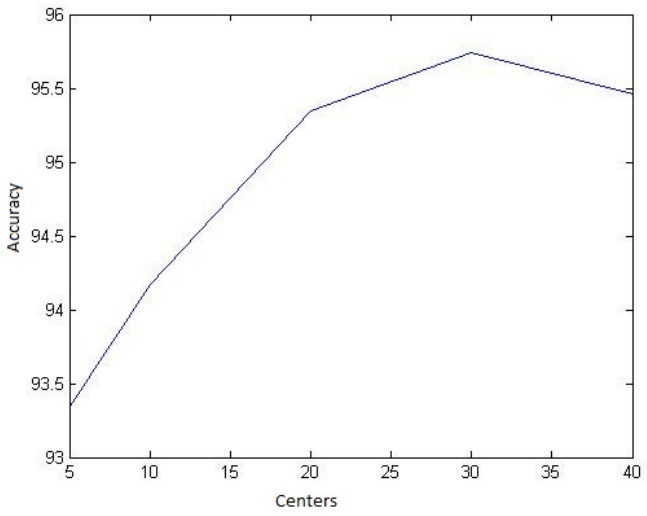

Figure 3: The variation of accuracy with r (Dataset: Gisette)

Observation: It was observed that accuracy in general increases with increase in r.

6.2. Multi Label Datasets

The following multi-label datasets were used in the project:

Dataset	Samples	Features	Labels
Image	2000	294	5
Scene	2407	294	6
Yeast	2417	103	14

Experiment 3:

1. The subsetsize was fixed at 50 and the number of centers r was varied are various multi-label performance metric were such as:
   (a) Coverage
   (b) Precision
   (c) Ranking Loss
   (d) Hamming Loss
   (e) One Error

   were calculated.
2. The process was repeated 10 times and the average values were plotted.
3. A reference line indicating the performance of LIFT algorithm was plotted on top of the obtained graphs for comparison.

Observation: It was observed that DISC algorithm with certain parameters performs better than LIFT. Also the performance of DISC improves with increase in r.

Experiment 4:

1. The process carried out was similar to experiment 3. Only this time r was kept fixed at 5 and the subsetsize s was varied.

Observation: It was observed that performance increases with decrease in subsetsize.

7. Plots

The following plots were obtained during the experiments:

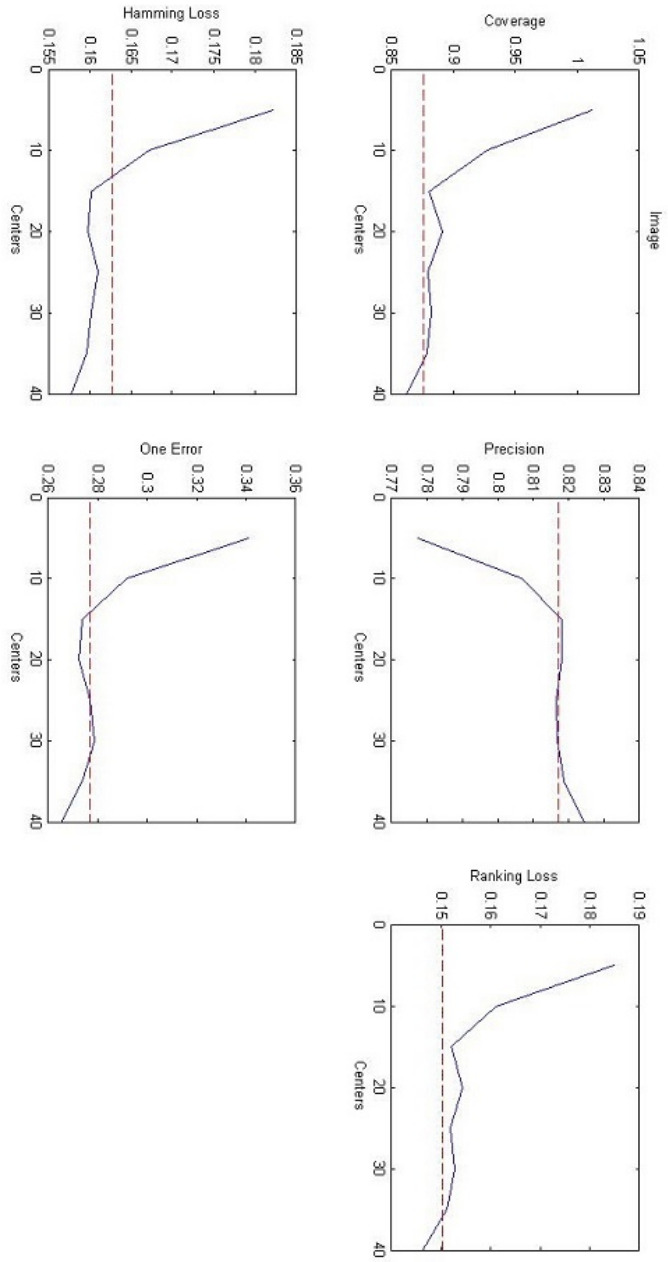

Figure 4: Variation of performance metrics with r (Dataset: Image).

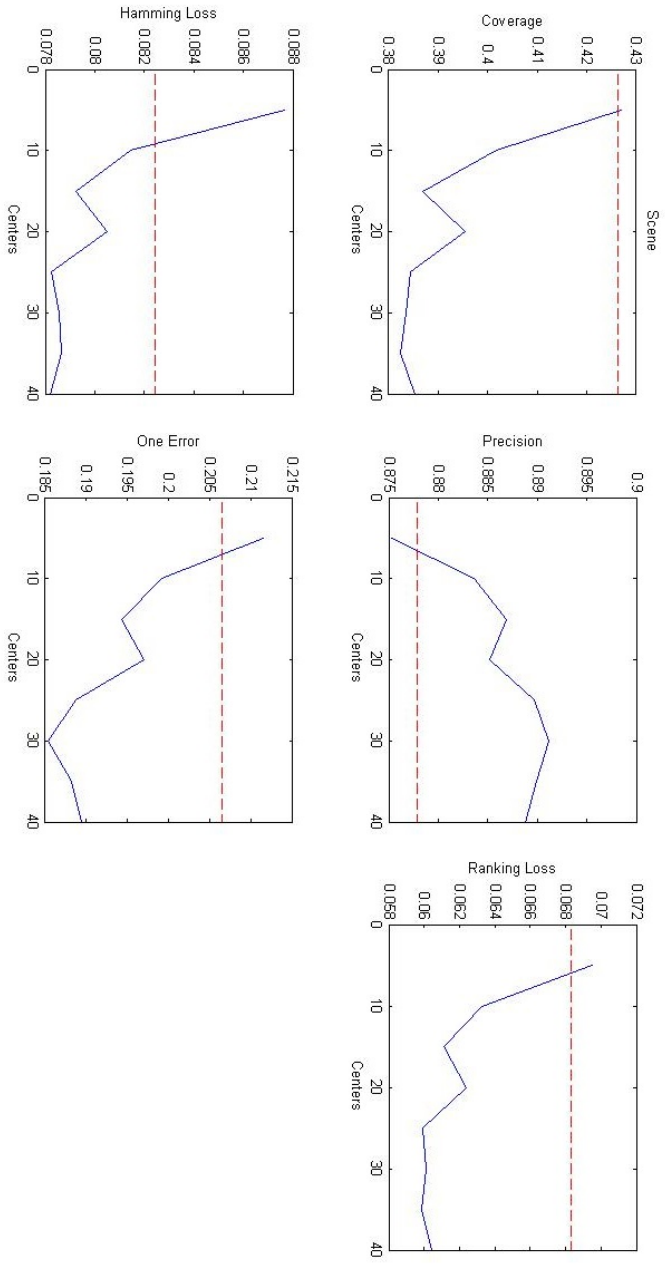

Figure 5: Variation of performance metrics with r (Dataset: Scene).

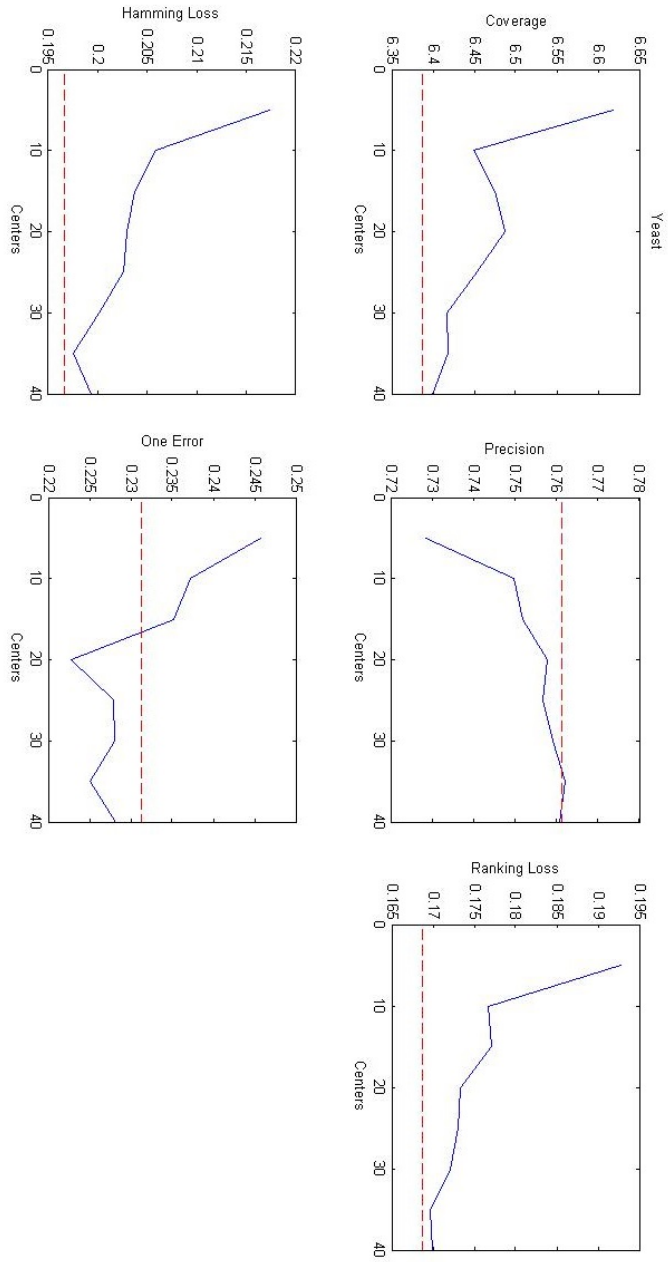

Figure 6: Variation of performance metrics with r (Dataset: Yeast).

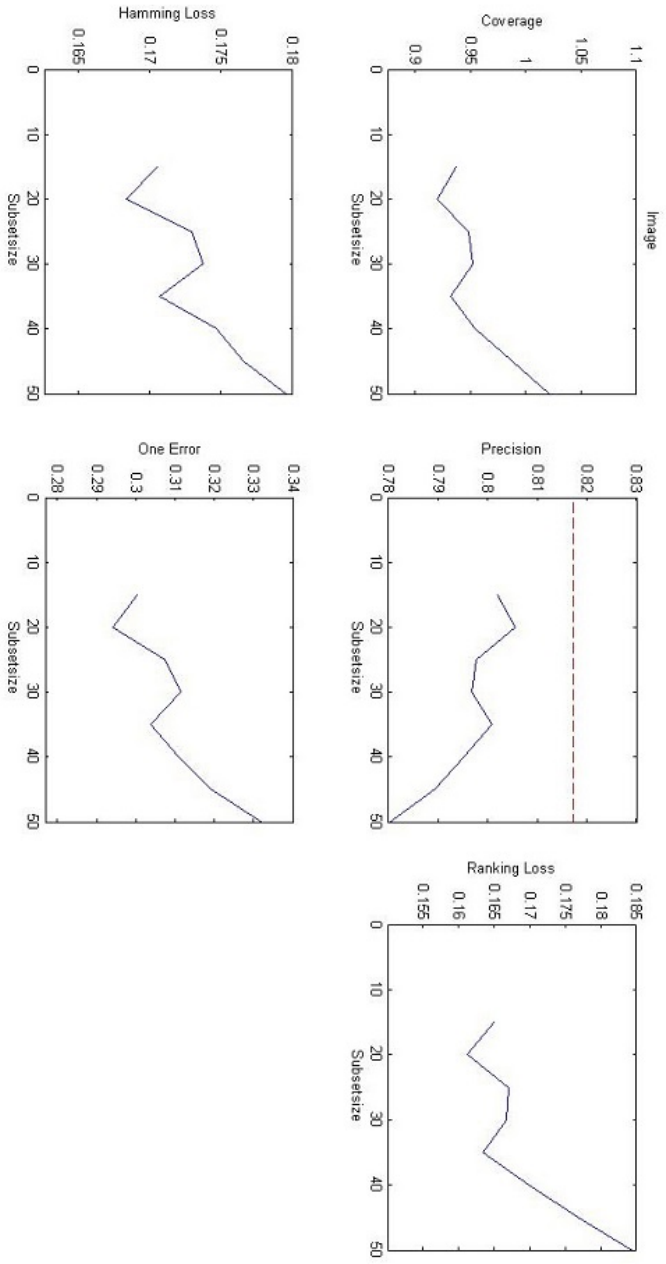

Figure 7: Variation of performance metrics with s (Dataset: Image).

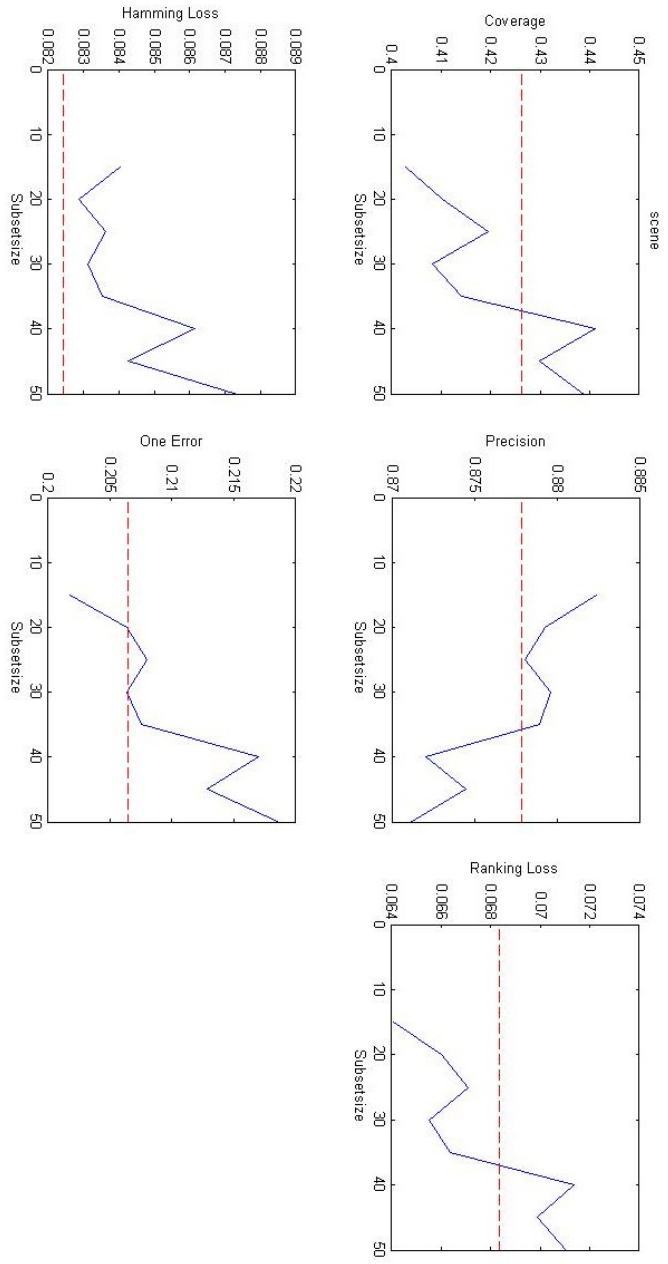

Figure 8: Variation of performance metrics with s (Dataset: Scene).

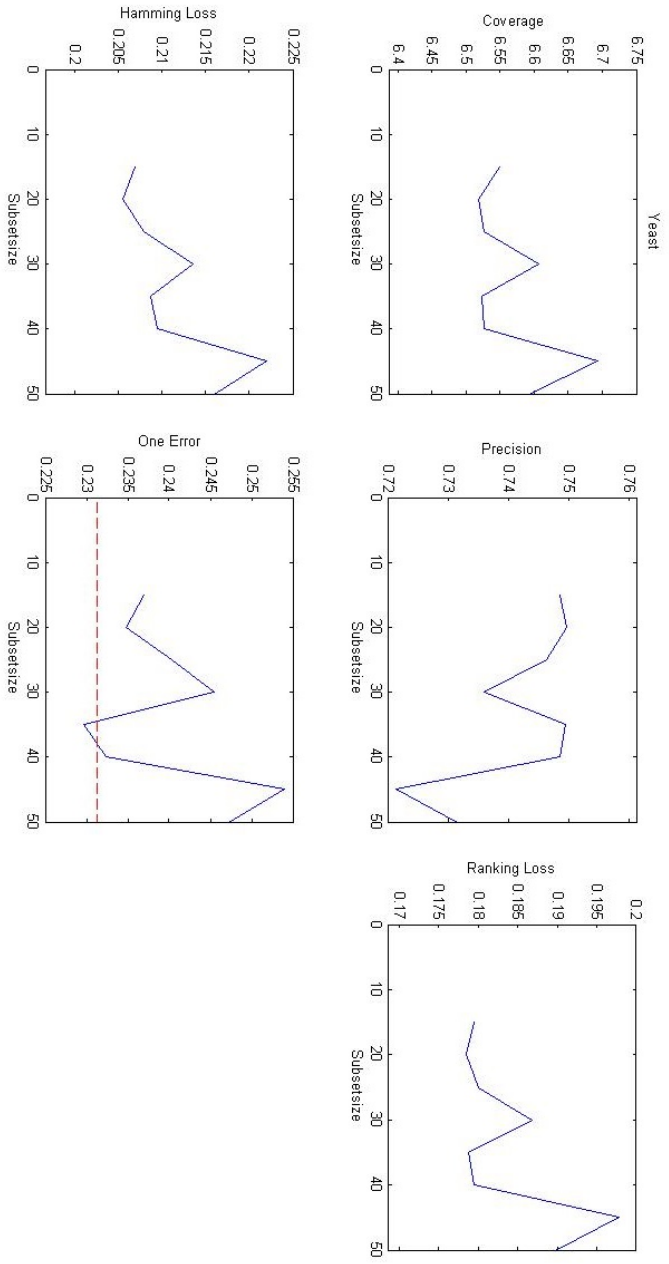

Figure 9: Variation of performance metrics with s (Dataset: Yeast).

8. Conclusion

In this project first a survey was conducted on existing dimensionality reduction techniques.Then, a novel algorithm called ‟DISC: Dimensionality reductIon Using Sample Clustering″ was proposed. Experiments conducted on various datasets validated its superiority against existing techniques with similar approach.