Fractal Dimension for Data Mining

Krishna Kumaraswamy
skkumar@cs.cmu.edu

Center for Automated Learning and Discovery
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213

Abstract

In this project, we introduce the concept of intrinsic "fractal" dimension of a data set and show how this can be used to aid in several data mining tasks. We are interested in answering questions about the performance of a method and also in comparing between the methods quickly. In particular, we discuss two specific problems – *dimensionality reduction* and *vector quantization*. In each of these problems, we show how the performance of a method is related to the fractal dimension of the data set. Using real and synthetic data sets, we validate these relationships and show how we can use this for faster evaluation and comparison of the methods.

Contents

1 Introduction

Many real life data sets have a large number of features, some of which are highly correlated. These correlated attributes contribute to the increase of complexity of any treatment that has to be applied to a data set (i.e., spatial indexing in a DBMS, density estimation, knowledge retrieval in data mining processes).

Also, these correlations often reveal interesting patterns and useful information hidden in the data. Data mining tasks such as dimensionality reduction, classification, clustering, learning patterns assist us in determining these patterns. These patterns can be seen as an indicator of the way the data points are spread in the data space.

A problem of interest is to perform these tasks efficiently and make inferences about the data quickly. In this project, we introduce the concept of "fractal" dimensions and show how we can use it to aid us in several data mining tasks. Fractal dimension is an estimate of the degrees of freedom of a data set. This we believe gives us an idea of the manner in which the data is spread in the data space. The spread of the data is usually related to the amount of information that we can obtain from the data. Also, the performance of a given data mining method is evaluated on the basis of the information it captures. However, implementation of most data mining methods is expensive and requires large computation time. We show how we can use the fractal dimension of a data set for making faster and better inferences.

In section 2, we introduce the problems addressed in this project. The concept of "fractal dimension" is introduced in section 3. In sections 4 and 5, we discuss two specific problems in greater detail and describe how we use the concept of "fractal" dimension in these problems. For each of these problems, we make some hypotheses and justify them using real as well as synthetic data sets. A brief discussion of the contributions of this project is listed in section 6 and the conclusions are presented in section 7.

2 Problem Definition and Motivation

The large number of correlated features increase the complexity of any treatment that has to be applied to a data set. This phenomenon is referred to as the *curse of dimensionality* [1] or as the *empty space phenomenon* [26]. However, a phenomenon which appears high–dimensional, and thus complex, can actually be governed by a few simple variables/attributes (sometimes called *hidden causes* or *latent variables*). A good dimensionality reduction method should be able to

ignore the redundant dimensions and recover the original variables or an equivalent set of them, while preserving the topological properties of the data set.

For the dimensionality reduction problem, we are interested in answering the following questions:

- Can we quickly determine the optimal dimension for the reduced space?
- Given two dimensionality reduction methods, how can we quickly compare the two methods?
- Can we do this in a way that is scalable to larger data sets?

Vector quantization is a lossy data compression method based on the method of *block coding*. Each data point is represented using a code vector. This is similar to the idea of clustering where the points in each cluster is represented by the centroid of the cluster. Several methods have been proposed for vector quantization and we are interested in answering the following questions:

- Does the performance of a vector quantizer depend on the data set?
- For a given data set, is there a limit to the performance of a vector quantizer?
- Can we do all this in a way that is scalable to larger data sets?

For each of the above methods, we show relations between the methods and the "fractal" dimension of the data set. Using real and synthetic data sets, we establish these relationships and show how we can use this for faster inferences about the data mining methods.

We first discuss the concept of fractals and that of "fractal" dimension of a data set in greater detail.

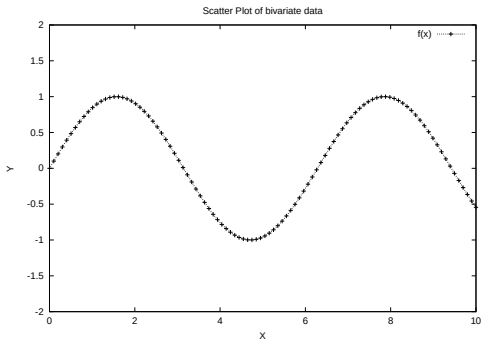
3 Fractal Dimension

Vector spaces may suffer from large differences in their **embedding dimensionality** and their **intrinsic dimensionality**. We define the **embedding dimensionality** $\mathbb{E}$ of a data set as the number of attributes of the data set (i.e. its address space). The **intrinsic dimensionality** $\mathscr{D}$ is defined as the real number of dimensions in which the points can be embedded while keeping the distances among them. It is roughly the degrees of freedom of the data set [5]. For example, a plane embedded in a 50-dimensional space has intrinsic dimension 2 and embedding dimension 50. This is in general the case in real data applications and it has led to attempts to measure the intrinsic dimension using concepts such as "fractal" dimension [9].

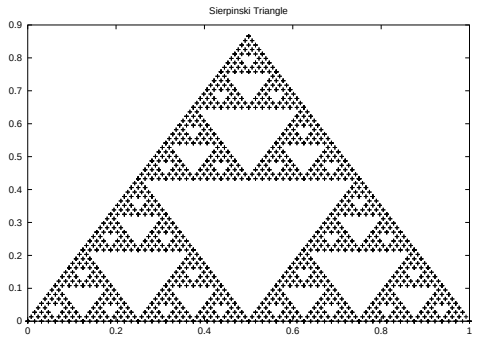
A fractal by definition is a self-similar point set. It consists of pieces which are similar to the original point set. For example, in Figure 1(b), we have the Sierpinski triangle. This consists of three smaller pieces that is similar to the original data set, each scaled down by a factor of 2.

Consider a perfectly self-similar object with r self-similar pieces, each scaled down by a factor s. The fractal dimension $\mathscr{D}$ for this object is given by

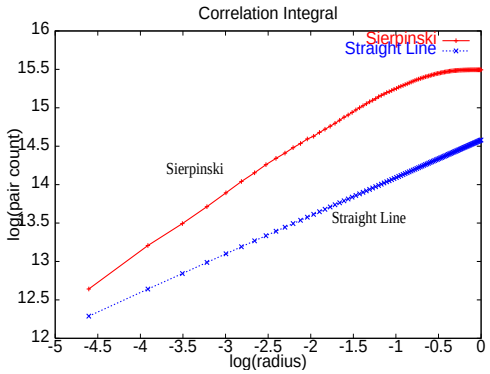
$$\mathscr{D} = \frac{ln\ r}{ln\ s} \tag{1}$$



(a) Sine wave



(b) Sierpinski Triangle



(c) Correlation Integral

Figure 1: Self-similar (fractal) data sets and their correlation integrals. Figure (c) shows the plot of the log number of pairs to the log of the radius. The sine wave has slope $\mathbb{D}$=1, and the Sierpinski triangle has slope $\mathbb{D}$=1.57 with behavior between a line($\mathbb{D} = 1$) and a plane($\mathbb{D} = 2$)

For example, the Sierpinski triangle has $r = 3$ and $s = 2$. Thus the fractal dimension of the Sierpinski triangle is given by $\mathscr{D} = ln\ r/ln\ s \simeq 1.58$.

In theory, any self-similar object should have infinitely many points as each self-similar piece is a replica of the original object. But in practice, we observe only a finite sample of the object.

For the finite data sets, we say that the data set is *statistically* self-similar on a given range of scales (r_{min}, r_{max}) on which the self-similarity assumption holds. A given finite data set is said to be statistically self-similar if it obeys the power law in the given range of scales.

To measure the intrinsic (fractal) dimension we use the slope of the *correlation integral* [9]. The *correlation integral* $C(r)$ for a data set is defined as:

$$C(r) \; = \; \#(pairs\ within\ distance\ r\ or\ less) \tag{2}$$

Then, the "fractal" dimension of the data set is defined as the exponent of the power law. Intuitively, the correlation fractal dimension indicates how the number of pairs (number of neighbors within a distance r) increases with increase in the distance

$$\text{number of pairs}(\leq r) \; \propto \; r^{\mathbb{D}} \tag{3}$$

Of course, for a *perfectly self-similar* object, $\mathscr{D} = \mathbb{D}$ [25]. In Figure 1, we have two data sets on a plane. The first is a simple sine curve and the second one is the *Sierpinski triangle*. Both data sets have an embedding dimension $\mathbb{E} = 2$. The correlation integral shows that the first data set has fractal dimension close to 1 as expected (one degree of freedom). However, the fractal dimension is not always an integer, nor does it always correspond to the degrees of freedom. For example, the *Sierpinski triangle* has a correlation "fractal" dimension of 1.57 which is more than that of the sine curve but less than that of a uniform distribution on a plane.

We use "fractal" dimension as a measure of the spread of the data and hence the intrinsic dimension of the data set. A fractal dimension of 0 means that there is no spread and a fractal dimension equal to $\mathbb{E}$ means that the spread is maximum.

4 Dimensionality Reduction

The problem we want to solve is the following:
Given a data set, and two dimensionality reduction methods, we want to find a tool to compare the performance of these methods in various data mining tasks. We want to do this comparison when the data mining task is **not yet** *specified.*

The goal is to find a tool to measure the success of a dimensionality reduction method in a way that is scalable to large data sets. Also, we want to be able to compare the performances of several dimensionality reduction methods without having to do cross-validation and without knowing the data mining task that is of interest. For any given dimensionality reduction method,

we also want to be able to find the optimal number of dimensions to which the input space must be reduced.

The performance of a method is measured using several data mining tasks (reconstruction, classification, etc) that measures that amount of information preserved by the dimensionality reduction method.

4.1 Survey

Dimensionality reduction is important in many types of applications. In *database applications*, query performances degrade as the dimensionality increases. In *classification applications*, the estimation in sparsely sampled high dimensional spaces affects the reliability of the obtained classification results. In *statistical applications,* multivariate density estimation based on the maximum likelihood (e.g., EM algorithm) is quite slow, result in many sub optimal solutions, and depend strongly on initial conditions. In each of these cases, a good dimensionality reduction method helps us in making these problems simpler.

Common approaches used for dimensionality reduction rely on parametric and non–parametric models. Here we describe some of these dimensionality reduction methods.

4.1.1 Principal Components Analysis

In Principal Components Analysis (PCA), the data is summarized as a linear combination of an orthonormal set of vectors [16]. The first principal component accounts for as much of the variability in the data as possible, and each successive component accounts for as much of the remaining variability as possible. This is the same as performing the singular value decomposition of the covariance matrix A as $U\ \Lambda\ V'$, where Λ is diagonal matrix of eigen-values, and U, V are orthonormal. We then set the lower eigen-values to zero to determine the reduced space. PCA is also referred to as Latent Semantic Indexing [6], Karhounen–Loeve transform [8] and Singular Value Decomposition in different applications.

4.1.2 Factor Analysis

Factor Analysis (FA) assumes that the data is a linear combination of real–valued uncorrelated Gaussian sources (i.e., latent factors) [13]. After the linear combination, each component of the data vector is also assumed to be corrupted with additional Gaussian noise u. The generative model is given by:

$$x = \Lambda z + u \quad , x \in \mathbb{R}^d,\ z \in \mathbb{R}^m,\ u \in \mathbb{R}^d,\ m \ll d. \tag{4}$$

where Λ is known as the *factor loading matrix*.

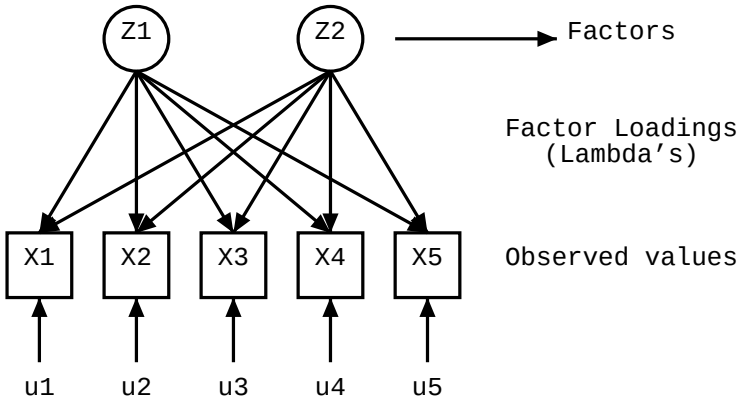


Figure 2: Simple path diagram for a factor analysis model

The goal of factor analysis is to find the factor loading matrix and determine the reduced space usually using *EM-algorithm* [13].

4.1.3 Artificial Neural Networks

Artificial Neural networks (ANN) can be used to implement some statistical methods as well [4]. The Self–Supervised MLP architecture (a.k.a. autoencoder) implements a mapping between two vector spaces using two layers of linear perceptrons with d input, m hidden units and d output trained to replicate the input in the output layer minimizing the squared sum of errors with back-propagation. This approach is called *self-supervised*, referring to the fact that during the training each output sample vector is identical to the input sample vector [7].

We use a five layer network with linear activation function for the input and output layers. We also have linear activation function for the middle layer. For the other two layers, we use a sigmoid activation function.

The process of dimensionality reduction consists of finding coding and decoding functions G and F that are (approximately) functional inverses of each other using back-propagation (see Figure 3) [1].

[1] We use the nodelib library to implement this network

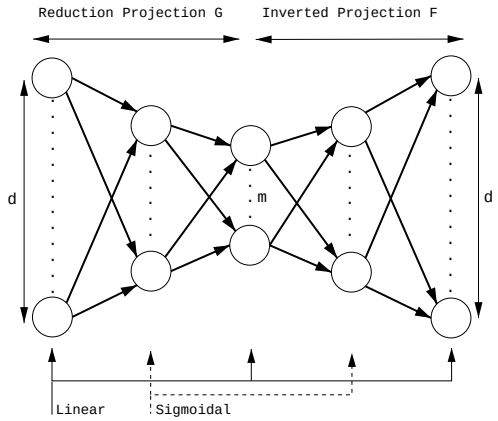


Figure 3: The Self–Supervised MLP architecture: d-inputs, d self replicated outputs and m hidden nodes

4.2 Proposed Conjecture & Approach

Consider a data set in a high-dimensional space. The goal of a dimensionality reduction algorithm is to determine an equivalent set of points in a lower dimensional space so that the pairwise distance between the points is more or less maintained. Consider a dimensionality reduction method that projects all the data points to the same point in the lower dimensional space (see Figure 4(b)). Clearly, this is not a good method as we lose all the information about the variation in the data points in the original space.

Is there a fast way to determine which method is better? We now propose our *MaxFD* hypothesis.

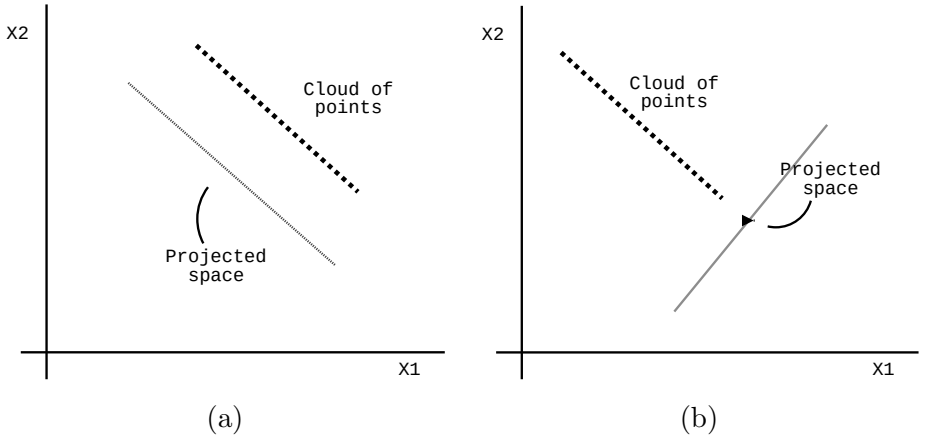


Figure 4: Projections of the same two dimensional space into a one dimensional space

Conjecture 1 *MaxFD*
With all other parameters being equal, a dimensionality reduction method which achieves higher "fractal" dimension in the reduced space is better than the rest for any data mining task.

A very informal argument is that if one dimensionality reduction method has a higher fractal dimension than another, then the latter method has probably lost some degrees of freedom.

We show that, using real data sets, this holds true, and therefore the fractal dimensionality of the resulting feature space can be used as an **a–priori** index for the final performance of various data mining tasks. For example, in Figure 1, the sine wave is less spread than the *Sierpinski Triangle* which is again less spread than an uniform distribution on the plane.

4.2.1 Experimental Setup

For practical applications, data preprocessing is often one of the most important stages in the development of a solution, and the choice of preprocessing steps can often have a significant effect on generalization performance [3]. In our data sets, different variables have typical values which differ significantly. However, the typical size of the inputs do not reflect their relative importance in determining the outputs. We perform a linear re-scaling of the input variables independently by subtracting their mean and scaling by the standard deviation.

We compute the *fractal dimension* of the data sets (see Section 4.3) using the correlation integral and the *classification errors* on the original data using all the attributes computed by a Decision Tree and a K-Nearest Neighbor classifier[2][3].

We then apply the described dimensionality reduction methods (PCA, Factor Analysis and Artificial Neural Networks) for each of the data sets for various values for the dimension of the reduced space. After applying dimensionality reduction methods, we measure their performance in terms of the reconstruction and classification powers of the reduced feature set. At the same time, we estimate the change in the topology structure of the reduced feature space using the fractal dimension of the projected data. We then show the relation between fractal dimension and the various dimensionality reduction methods in terms of their performance in various data mining tasks.

[2]The DT classification is based only on the information given from a single attribute in predicting the class, while the KNN classification uses the topology structure to determine the class.

[3]We use C4.5 to build our decision tree for classification task [24]

4.3 Experimental Results

4.3.1 Data sets

In order to validate our hypothesis, we apply the dimensionality/feature reduction methods to a few real data sets. We scale the data according to the method proposed in the previous section and compute the fractal dimension of the original and the re-scaled data sets using the correlation integral. We use the following data sets for our analysis:

- *Protein Images:* 862 records and 84 attributes showing the images of 10 sub-cellular patterns in HeLa cells described in [20];
- *Motion Capture Data:* 97 records with 79 attributes measuring the various angles of joints of a human actor during running motion;
- *Basketball data:* 459 records with 47 numerical attributes showing basketball statistics from the 1991-92 NBA season, including minutes played, field goals, rebounds, and fouls used in [17];
- *Hockey data:* 871 players with 14 attributes like games played, goals scored, assists made, points scored and classes indicating their positions (Center, Right, Left, Defense, Goal) used in [22].

Here, the *Protein Images* and *Hockey* data sets are labeled data sets. For these data sets, we look at the classification errors in addition to the reconstruction error.

We vary the number of features for the reduced space (k) and perform dimensionality reduction for each of these values. For each of the reduced spaces, we compute the fractal dimension using the correlation integral to show the correlation between the data mining capabilities and the fractal dimension. To evaluate the performance of the dimensionality reduction method, we compute the *reconstruction* and *classification errors*[4]. We repeat this for each of the data sets mentioned above. According to our *MaxFD* conjecture, the higher the fractal dimension of the reduced space, lower is the error. We perform experiments to answer the following questions:

1. For a given dimensionality reduction method, we want to determine the optimal dimension of the reduced space.
2. Given several dimensionality reduction methods, we want a tool to compare their performances.
3. We want to do all this in a way that is scalable to large data sets.

[4]We use 5–fold cross–validation and t–test to verify the statistical significance of the different results.

4.3.2 MaxFD and target dimensionality ($\hat{k}$)

Notice that for each data set and the different dimensionality reduction methods, the reconstruction and classification errors decrease with increase in the number of dimensions in the reduced space (see Figures 5, 6). Also notice that these errors stabilize after a certain stage. The fractal dimension of the data set also increases with the increase in the number of dimensions and stabilizes when the errors stabilize. Each of the plots are *dual–axis plots*. Notice that the left Y axis shows the errors and the right Y axis shows the fractal dimension. The X axis shows the different number of features retained.

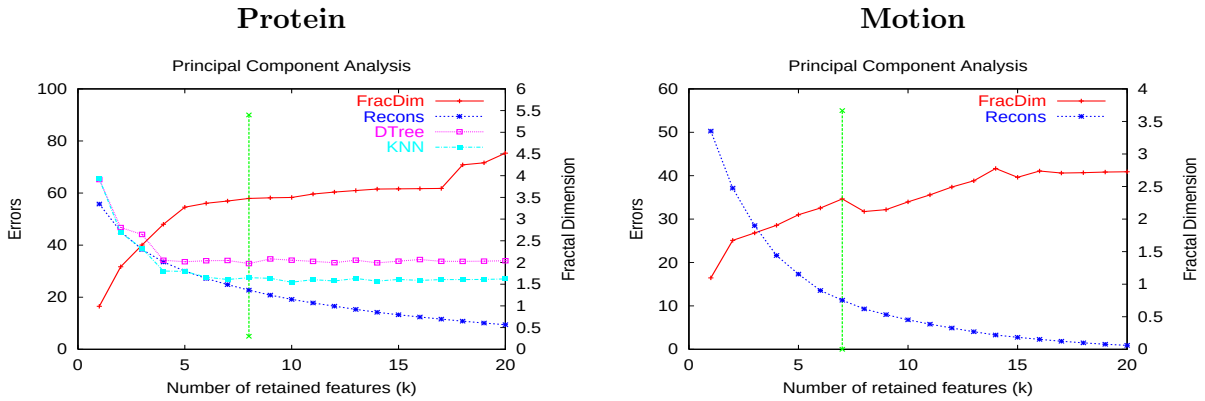
The vertical line in each of these plots indicates the number of retained features k after which there is no significant change in the fractal dimension of the reduced space. This point indicates the point of flattening of the curves.

Once the fractal dimensions stabilize, notice that the measures of error also stabilize. This indicates that once the fractal dimension of the reduced space stabilizes, then there is no significant increase in the improvement of the reduced space. Thus, we can use the fractal dimension as an indicator for determining the optimal number of dimensions. The point of flattening of the fractal dimension can then be used as the target dimension for the reduced space ($\hat{k}$).

4.3.3 MaxFD and choice of dimensionality reduction method

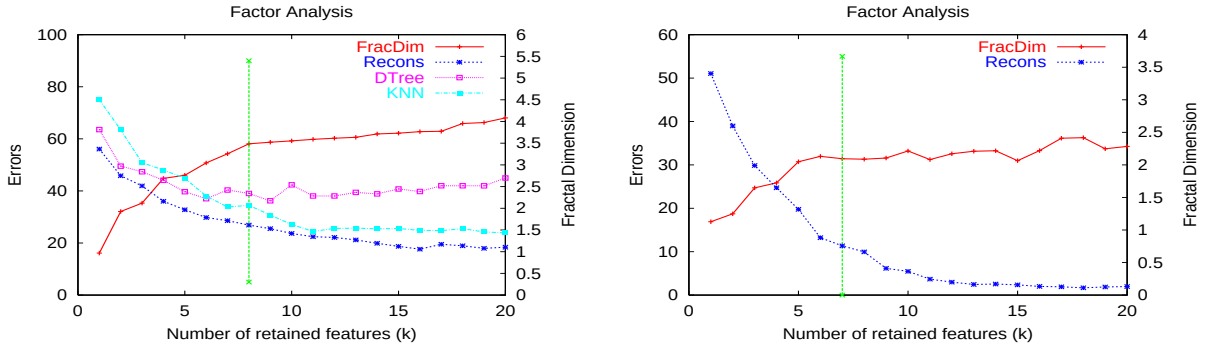
Once we have determined the optimal number of features ($\hat{k}$) to retain, we can then look at the relation between the error and the fractal dimension of the reduced space based on the various methods. For the sake of comparison, we include two additional methods for reducing the number of dimensions – namely *High Variance* and *Random*. We pick $\hat{k}$ variables with the highest variances before scaling for the *High Variance* method and pick $\hat{k}$ attributes randomly for the *Random* method. Figure 7 shows the plot of the various errors against the fractal dimension for a fixed dimension of reduced space and different dimensionality reduction methods. The dotted line has been drawn manually to serve as a visual aid. The results from the original data set with all attributes is also shown. Notice from the plots that the good dimensionality reduction methods gives a lower error and a higher fractal dimension. The higher the fractal dimension of the reduced space, the lower are the errors of our data mining tasks among the different dimensionality reduction methods. This gives us a way of choosing the best dimensionality reduction method for a fixed number of dimensions in the reduced space.

Why does it work? MaxFD tries to find the real degrees of freedom of the data set. This in-



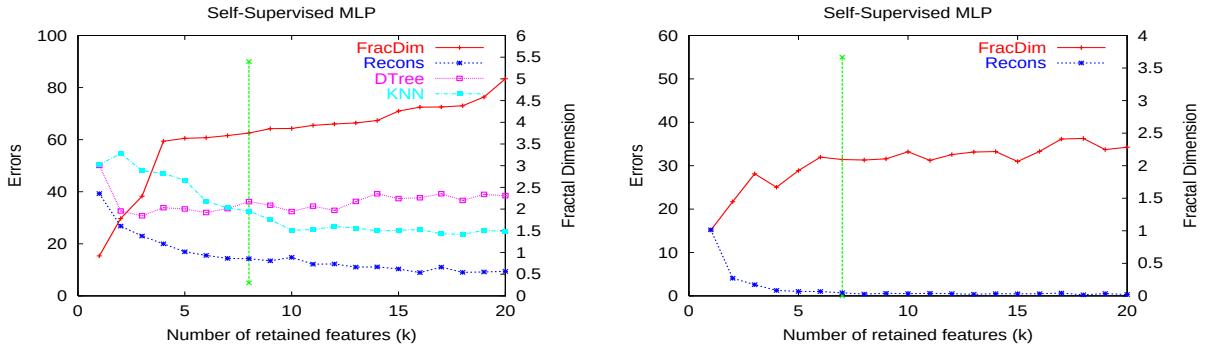
(a1) (a2)

(a) Principal Component Analysis



(b1) (b2)

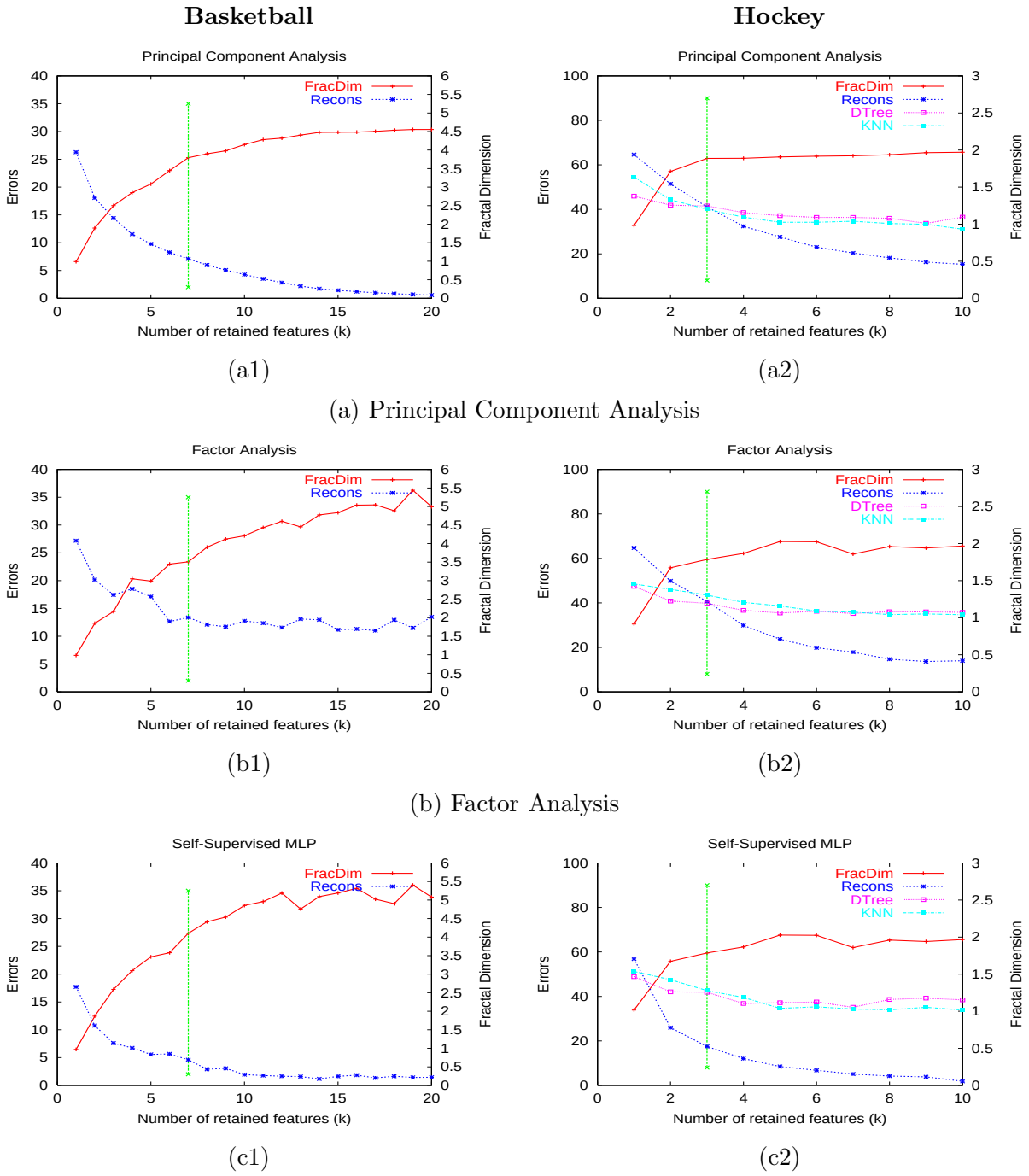
(b) Factor Analysis



(c1) (c2)

(c) Artificial Neural Networks

Figure 5: *Dual-axis plot:* plot of errors (reconstruction, decision tree classification & KNN classification) and fractal dimension (FracDim) against the number of retained features using different methods for the Protein and Motion data sets. Note that the errors stabilize when fractal dimension $\mathbb{D} = 3.5$ and 2.5 for Protein and Motion data sets, respectively. Here, the optimal dimension ($\hat{k}$) of the reduced spaces are 8 and 7 for the protein and motion data sets respectively as indicated by the vertical lines



(a1) (a2)

(a) Principal Component Analysis

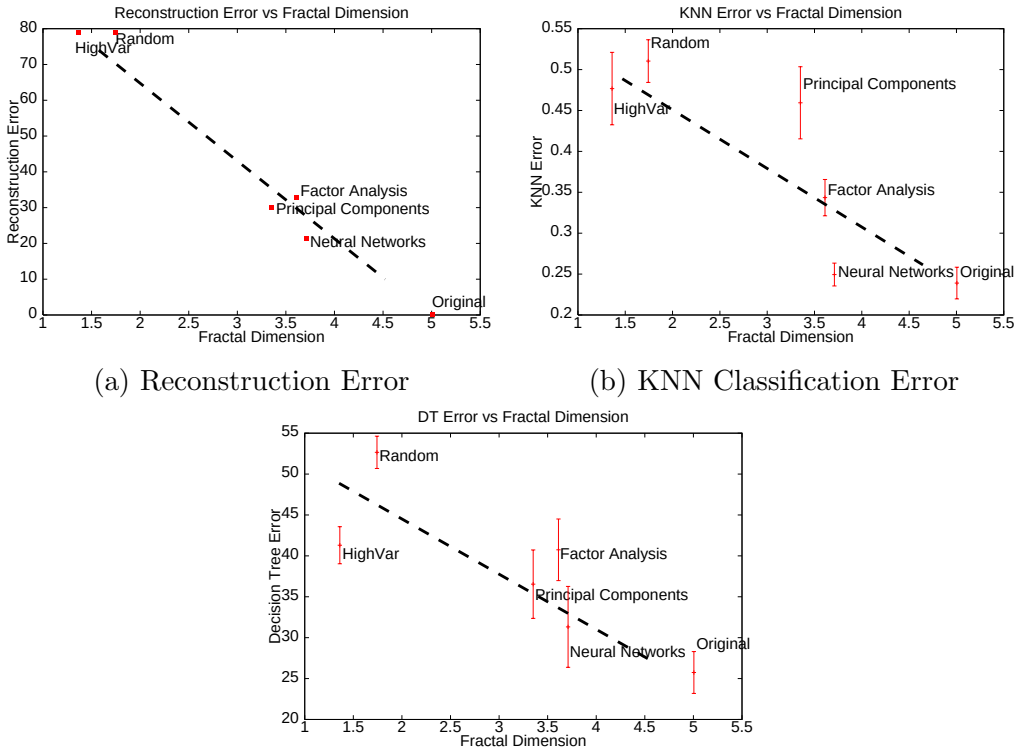
(b1) (b2)

(b) Factor Analysis

(c1) (c2)

(c) Artificial Neural Networks

Figure 6: *Dual-axis plot:* plot of errors (reconstruction, decision tree classification & KNN classification) and fractal dimension (FracDim) against the number of retained features using different methods for the Basketball and Hockey data sets. Note that the errors stabilize when fractal dimension $\mathbb{D} = 4.5$ and 2.0 for Basketball and Hockey data sets, respectively. Here, the optimal dimension ($\hat{k}$) of the reduced spaces are 7 and 3 for the basketball and hockey data sets respectively as indicated by the vertical lines



(a) Reconstruction Error

(b) KNN Classification Error

(c) Decision Tree Classification Error

Figure 7: Comparison of several dimensionality reduction methods for a fixed embedding dimension (5) for the Protein data set. Notice that higher the fractal dimension, lower is the error.

tuition is best explained in Euclidean data sets where the fractal dimension measures the degrees of freedom of the data set. *MaxFD* retains the natural degrees of freedom in the data set. A bad projection (as in Figure 4(b)) reduces the fractal dimension of the reduced space and this is picked up by *MaxFD*.

Why should we use this? Evaluating a dimensionality reduction technique is usually an expensive task depending on the type of data mining task we are interested in. This usually involves cross-validation to effectively evaluate the performance of the reduced space. However, computation of MaxFD scales linearly to the number of observations in the data set [27].

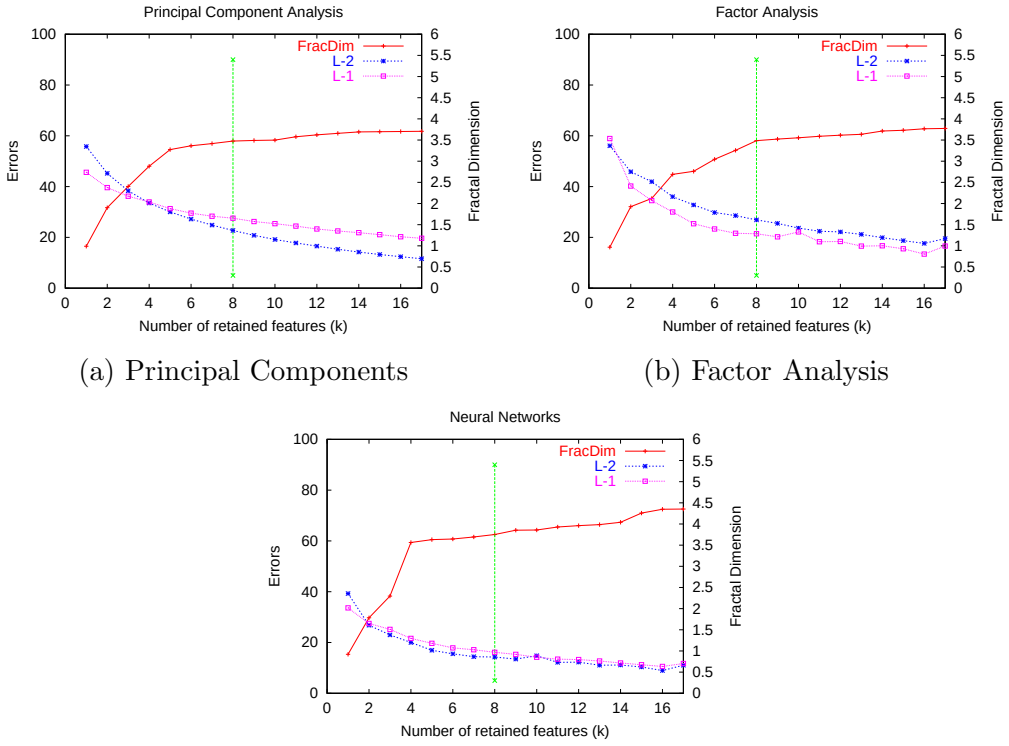
How can we use it? We showed that the fractal dimension of the data set seems strongly correlated with the various data mining tasks and can be used to measure the performance of the dimensionality reduction. We can use this information to find

- A better method to determining the optimal number of dimensions of reduced space (better

than using 95% of variance as the criterion that SVD/KL recommends [10]).

- For a given dimensionality reduction method, MaxFD helps us determine the target dimensionality of the reduced space
- For a fixed dimension for the reduced space, MaxFD helps us determine the method that gives the best results for various data mining tasks
- A faster and scalable way to determine the performance of the dimensionality reduction method

4.3.4 Other error metrics



(a) Principal Components (b) Factor Analysis

(c) Neural Networks

Figure 8: Comparison of different error metrics for the reconstruction error in the protein data set. Notice again that we have a dual-axis plot with the different error metrics on one Y-axis and the fractal dimension on the other Y-axis.

In the experiments above, we used squared error loss as our error metric for computing the reconstruction error. However, for different problems, we might be interested in using different

error metrics.

A good dimensionality reduction method should be able to preserve the information that is stored in the data set. This is independent of the type of distance metric we use or the type of errors. As a result, we expect the *MaxFD* conjecture to hold true even with different metrics. For the protein data set, we show results using the L_1 and L_2 distance metrics. [5]

Notice than even with the L_1 metric, the reconstruction errors from each of the different dimensionality reduction methods stabilize after some time. Also notice that when these errors stabilize, the fractal dimension of the reduced space also stabilized (see Figure 8). This further leads us to believe that the *MaxFD* conjecture might hold even for other distance metrics. However, further experiments are needed to make a stronger conclusion in this regard.

5 Fractal Dimension and Vector Quantization

Vector Quantization is a lossy data compression method based on the principles of *block coding*. Each observation in the data set is represented by a fixed number of representative points that are chosen to reduce the loss. In a single dimension this is called *scalar quantization* and the idea is similar to "rounding off" (say to the nearest integer). For example, if we represent 16 bit values by the 8 most significant bits, then we get an approximation of the original data at the expense of precision.

A vector quantizer maps a data set in a n-dimensional data space into a finite set of vectors $Y = \{y_i, i = 1, 2, \ldots k\}$. Each vector y_i is called a code vector or a codeword (representative element). The set of all codewords is called a codebook. Associated with each codeword y_i is a nearest neighbor region called Voronoi region containing all points that are closer to y_i than to the other codewords.

Vector Quantization is used in compression algorithms for data transmission or storage. Each input vector can be associated with an index of a codeword and this index may be transferred instead of the original vector. The index can then be decoded to get the codeword that it represented.

The problem we are interested in is as follows:

[5] With the L_3 or L_∞ norms, there is a difference in magnitude of the errors. As a result, the results cannot be easily compared to the results from the $L - 2$ norm.

1. Does the performance of a vector quantizer depend on the data set? If so, how?

2. Is there a limit to the performance of the Vector Quantization algorithm for a given data set?

3. Can we determine all this in a fast and scalable way?

5.1 Survey

5.1.1 Quantization

Quantization is the discretization of a continuous-alphabet source [12, 23]. The *source* refers to the mechanism that generates the data to be encoded and usually is modeled by some particular probability distribution.

Scalar Quantization: *Scalar* (one dimensional) quantization is a data compression technique that approximates a source symbol by its closest (minimum distortion) representative from a predetermined finite set of allowed values (codewords) stored in a codebook. The *distortion*, d, is a measure of overall quality degradation.

It is an assignment of a non-negative cost $d(X, \tilde{X})$, associated with quantizing any input value X with a reproduction value $\tilde{X}$. Given such a measure we can quantify the performance of a system by an average distortion $D = Ed(X, \tilde{X})$ between the input and the final reproduction. In practice, the overall measure of performance is the long term sample average or time average

$$\overline{d} = \lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} d(X_i, \tilde{X}_i) \tag{5}$$

where $\{X_\nu\}$ is a sequence of values to be encoded [12]. If the process is stationary[6] and ergodic[7], then with probability one the above limit exists and it is equal to the statistical expectation, i.e., $\overline{d} = D$. The most common distortion function, is the mean squared error $E[(X - \tilde{X})^2]$, where $\tilde{X}$ is the final reproduction of any input value X. This is the distortion measure that we will adopt.

In scalar quantization the index of the closest codeword is determined during encoding, transmitted or stored, and used to look up the codeword at the decoder. This is done instead of sending the actual value in order to save bandwidth (for transmission) or space (for storage).

[6] A random process is stationary if the probabilities of all events are not affected by time shifts.

[7] A random process is ergodic if, with probability one, all the statistical averages can be determined from a single sample function of the process. In effect, the random process is ergodic if time averages obtained from a single realization are equal to the statistical averages.

We define an N-point scalar quantizer Q as a mapping $Q : \mathcal{R} \to \mathcal{C}$ where $\mathcal{R}$ is the real line and $\mathcal{C} = \{y_1, y_2, \ldots, y_N\} \subset \mathcal{R}$ is the codebook of size N. The output values, y_i, are referred to as *reproduction values*.

Every quantizer can be viewed as the combined effect of two successive operations (mappings), an *encoder*, $\mathcal{E}$, and a *decoder*, $\mathcal{D}$. The encoder is a mapping $\mathcal{E} : \mathcal{R} \to \mathcal{I}$ where $\mathcal{I} = \{1, 2, \ldots, N\}$, and the decoder is the mapping $\mathcal{D} : \mathcal{I} \to \mathcal{C}$. Thus, if $Q(x) = y_i$, then $\mathcal{E}(x) = i$ and $\mathcal{D}(i) = y_i$. With these definitions we have $Q(x) = \mathcal{D}(\mathcal{E}(x))$.

Vector Quantization: *Vector* quantization (VQ) is an extension of the previous scheme to a block or vector of source symbols and vector codewords in the codebook (see [12, 21, 15] for a complete treatment). The advantage of VQ over scalar quantization can be determined theoretically for asymptotically large vector dimension [2, 11]. In fact, the optimality of VQ is not limited to asymptotic considerations; for a statistically specified random vector of any given dimension, there exists no better way to quantize this vector with a given number of bits than with VQ [12]. In fact, VQ is superior to scalar quantization because it exploits the possible linear dependence (correlation) and nonlinear dependence that exists between the components of a vector. However, VQ still gives superior performance over scalar quantization even when the components of a random vector are statistically independent of each other! This follows from the basic Shannon source coding theorems [11, 2, 14]. It is due to the extra freedom in choosing the multidimensional quantizer cell shapes in VQ.

A vector quantizer Q of dimension k and size N is a mapping from a vector (or a "point") in k-dimensional Euclidean space, $\mathcal{R}^k$, into a finite set $\mathcal{C}$ containing N output or reproduction points, called *codewords*. Thus, $Q : \mathcal{R}^k \to \mathcal{C}$, where $\mathcal{C} = \{y_1, y_2, \ldots, y_N\}$ and $y_i \in \mathcal{R}^k$ for each $i \in \mathcal{J} \equiv \{1, 2, \ldots, N\}$. The set $\mathcal{C}$ is the codebook of size N. Associated with every N–point vector quantizer is a partition of $\mathcal{R}^k$ into N regions or cells, R_i for $i \in \mathcal{J}$. The ith cell is defined by

$$R_i = \{x \in \mathcal{R}^k : Q(x) = y_i\}, \tag{6}$$

For a given codebook, $\mathcal{C}$, the optimal partition cells satisfy the following Nearest Neighbor Condition:

$$R_i \subset \{x : d(x, y_i) \leq d(x, y_j); \forall j\} \tag{7}$$

that is

$$Q(x) = y_i \text{ only if } d(x, y_i) \leq d(x, y_j) \text{ all } j \tag{8}$$

Also, for given partition regions $\{R_i : i = 1, \ldots, N\}$ the optimal reconstruction vectors satisfy the Centroid Condition: $y_i = cent(R_i)$ where the centroid, y, of a set $R \in \Re^k$ is:

$$cent(R) = arg\ min_{y \in R}(E[d(X, y)|X \in R]) \tag{9}$$

For the squared error measure, the centroid of a set R is the arithmetic average

$$cent(R) = \frac{1}{|R|} \sum_{i=1}^{|R|} x_i \tag{10}$$

for $R = \{x_i : i = 1, \ldots, |R|\}$, where $|R|$ is the cardinality of the set R. The reader is encouraged to see [12] for the proofs of the above statements.

An efficient and intuitive codebook design algorithm for vector quantization, the Generalized Lloyd Algorithm [19, 18] (GLA), produces a "locally optimal" codebook from a training sequence, $\mathbf{T}$, typical of the source to be coded. A quantizer is *locally optimal* if every small perturbation of the code vectors does not lead to a decrease in the average distortion [12]. If we have a codebook that satisfies both necessary conditions of optimality, it is widely believed that this solution is indeed locally optimal, although no general theoretical derivation of this result has ever been obtained [12].

There is no better way to quantize a single vector than to use VQ with a codebook that is optimal for the probability distribution describing the random vector. However, the codebook design and the VQ encoding (i.e., search in the codebook for optimal vector) for unconstrained VQ are computationally very expensive especially for higher rates, large vector dimensions and codebook sizes. It is rather common to use unconstrained VQ with modest vector dimensions and codebook sizes. Several techniques have been proposed to reduce the computational complexity by applying various constraints to the structure of the VQ codebook. As a result, higher vector dimensions and larger codebook sizes become feasible without hitting complexity barriers. These methods usually compromise the performance achievable by the unconstrained VQ but provide very useful trade-offs between performance and complexity. Moreover, they can often be designed for larger dimensions and rates.

Generally, the performance of VQ can only increase as the dimension k of the vector increases. This is due to exploiting longer term statistical dependency among the signal samples (and the extra freedom in choosing higher dimensional quantizer cell shapes). The storage and search complexity are both proportional to $k \cdot N$, where N is the codebook size. Note that $N = k \cdot 2^{r \cdot k}$ where $r = log(N)$ is the rate. Thus, both time and space grow exponentially with the dimension.

When imposing a structural constraint on the codebook the codevectors cannot have arbitrary locations as points in k-d space but are distributed in a restricted manner that allows a much easier search for the nearest neighbor. In other cases a search procedure may not find the nearest neighbor but one of the approximately nearest neighbors.

5.2 Our Proposal

Let k be the number of codewords (representative elements) for Vector Quantization and let $E(k)$ be the root mean squared error (RMSE) obtained. Let $\mathscr{D}$ be the fractal dimension of the data set.

Theorem 1 *FD and VQ*
For a perfectly self-similar data set, the error $E(k)$ is related to the fractal dimension $\mathscr{D}$ as follows:

$$log\ E(k)\ =\ const - \frac{1}{\mathscr{D}} \cdot log\ k \tag{11}$$

Proof: Here, *log k* is called the rate for Vector Quantization. We want to show that the slope of the log error and the rate is inversely proportional to the fractal dimension of the data set.

Let N be the number of points in the self-similar data set. Let $E(N, k)$ be the error we get when we use vector quantization with k codewords. Also, let the data have r similar pieces, each scaled down by a factor s. For example, in the Sierpinski triangle, we have three self-similar pieces, each one half the size of the original. Thus, we have $r = 3$ and $s = 2$. For further simplification, assume that the number of codewords k grows as a power of r, i.e. $k = r^{k'}$ for some k'.

If $k = 1$, then we have only one group with all the data points. Now, if $k = r$, then we would have one codeword for each of the r similar pieces. Each of the r similar pieces would have N/r points and the distances between the points are scaled down by a factor of s compared to the original data set. Every time we increase the number of codewords by a factor r, they are distributed uniformly over the r similar pieces.

Thus, at each step, we get r scaled down replications of the original object and as a result, the total error can be written as a multiple of the error from one of the individual pieces. The error from each of the pieces are scaled down by a factor s.

We can thus rewrite the expression for the error as follows:

$$E(N, k) \quad = \quad E(N, r^{k'})$$

$$
\begin{aligned}
&= r \cdot E\left(\frac{N}{r}, r^{k'-1}\right) \cdot \frac{1}{s} \\
&= r \cdot \frac{1}{r} \cdot E\left(N, r^{k'-1}\right) \cdot \frac{1}{s} \\
&\vdots \\
&= E(N,1) \cdot \frac{1}{s^{k'}} \\
&= const \cdot \frac{1}{s^{k'}}
\end{aligned}
$$

Since $k = r^{k'}$, we have $log\ k = k' \cdot log\ r$ and thus the above equation becomes

$$
\begin{aligned}
log\ E(N,k) &= const - k' \cdot log\ s \\
&= const - \frac{log\ k}{log\ r} \cdot log\ s \\
&= const - \frac{1}{\frac{log\ r}{log\ s}} \cdot log\ k \\
&= const - \frac{1}{\mathscr{D}} \cdot log\ k
\end{aligned}
$$

Hence the proof of the theorem. □

Lemma: The constant is given by $const = log \sum_{i=1}^{N} (x_i - \bar{x})^2$ which is called the zero-rate information in Vector Quantization terminology.

Conjecture 2 *FD and VQ*
For a statistically self-similar data set, the same relation holds with

$$log\ E(k) = const - \frac{1}{\mathbb{D}} \cdot log\ k \tag{12}$$

Notice that the proof of the result relies only on the fact that the data set is self-similar. If we have a different error metric, only the constant changes and we get the same result. The constant in this case would be changed appropriately.

5.3 Experimental Results

For any perfectly self-similar object, our result clearly holds true. We test our results on a few synthetic and real data sets. We perform experiments to establish our result on synthetic as well as real data sets. Also, we want to estimate the performance of a vector quantizer without having to implement the algorithm.

For each data set, we run the vector quantization algorithm with different values of k and note the errors for the same. We then look at the plot of the log error against the rate to determine if our result holds true or not.

5.3.1 Data sets

To test our result, we use the following real and synthetic data sets:

- Sierpinski Triangle: A sample of 10,000 points from the Sierpinski Triangle;
- Koch Curve: A sample of about 16,000 points from the Koch curve;
- Diagonal Data: A random set of 10,000 points lying on a straight line;
- Montgomery County Data: A real data set of points from Montgomery county;
- Long Beach County Data: A real data set of points from Long Beach county.

For each of the data sets, we perform vector quantization with varying sizes of the codebook and compute the errors. We then plot the errors and the codebook size in the log-scale. We also show the plot of the cluster centroids determined by the vector quantizer. Having obtained the errors from the vector quantizer, we then fit a line in the log-scale to determine the slope. From this we estimate the fractal dimension of the data set (using the above theorem). We also estimate the fractal dimension using the correlation integral. We compare these to ensure that our result is indeed true.

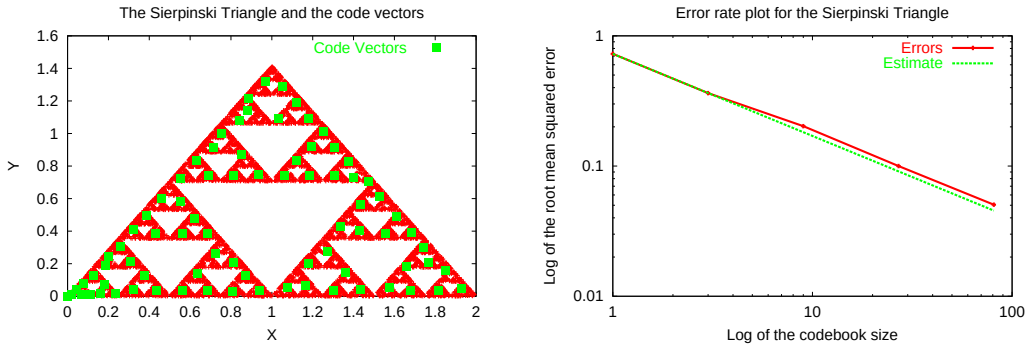


Figure 9: The Sierpinski triangle and the cluster centroids and the error rate plot. The error plot has slope -0.60316 and gives fractal dimension of 1.65792

For each of the plots, notice that the error rate plot gives us a straight line in the log-scale (see Figures 9, 10, 11, 12 and 13). Also notice that the clusters centroids (codewords) are spread according to the spread of the data (i.e. there are more cluster centroids near the dense areas and fewer in sparse areas). The fitted line for the error plot seems to be very close to the line estimated using our result. This suggests that our hypothesis is true. To verify this, we compute the slope and compare it to the estimate of the fractal dimension obtained using the correlation

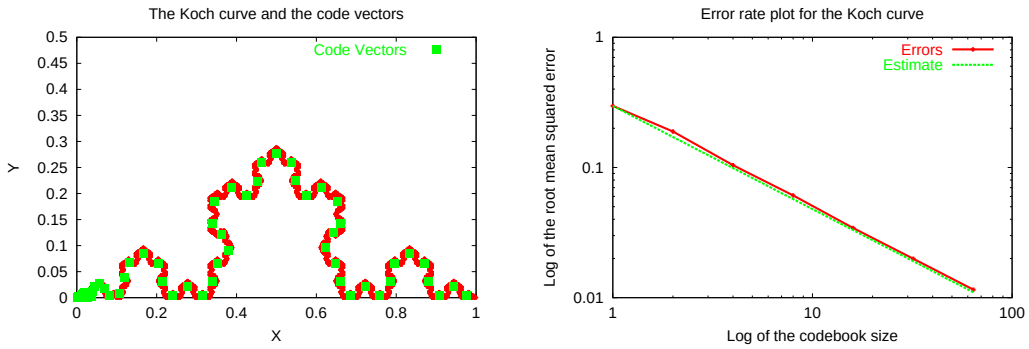


Figure 10: The Koch curve and the cluster centroids and the error rate plot. The error plot has slope -0.75172 and gives fractal dimension of 1.33028

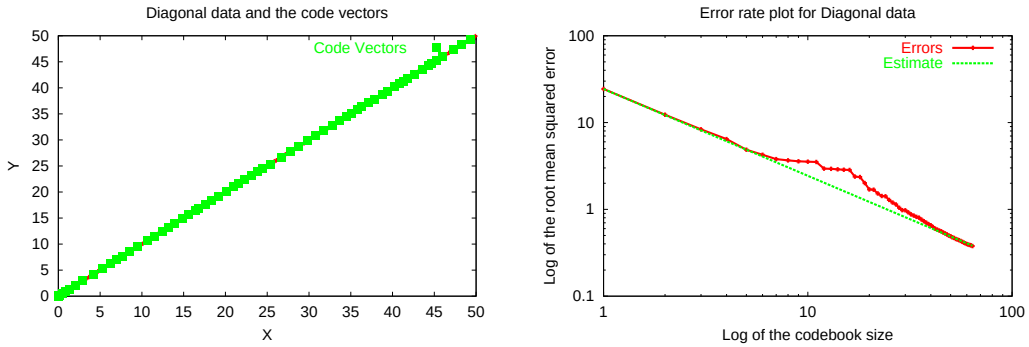


Figure 11: The synthetic diagonal data and the cluster centroids and the error rate plot. The error plot has slope -1.00597 and gives fractal dimension of 0.99406

integral. Notice that the estimate of the fractal dimension from the slope of the error plot and the fractal dimension computed from the slope of the error plot are almost the same (see Table 1). This suggests that our result holds true on real as well as synthetic data sets.

As a visual aid, we estimate the line from the fractal dimension of the data set to compare with the error obtained from the Vector Quantization method. Notice that the line estimated using the above result is in excellent agreement with the results obtained using a vector quantizer. This confirms that our result is indeed true.

Also, we note down the time taken for performing the vector quantization and for estimating the fractal dimension using the correlation integral. A simple comparison of the times reveal that it is far easier to estimate the fractal dimension than performing the vector quantization. This it is easier to determine the properties of a vector quantizer using the fractal dimension instead of

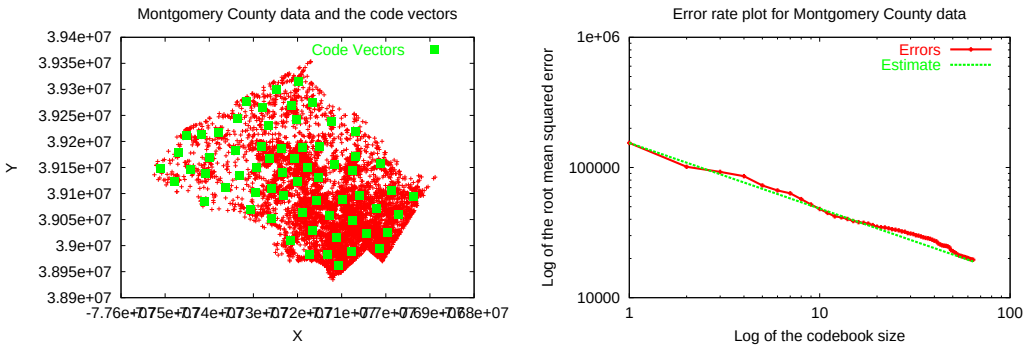


Figure 12: The Montgomery county data and the cluster centroids and the error rate plot. The error plot has slope -0.54233 and gives fractal dimension of 1.80637

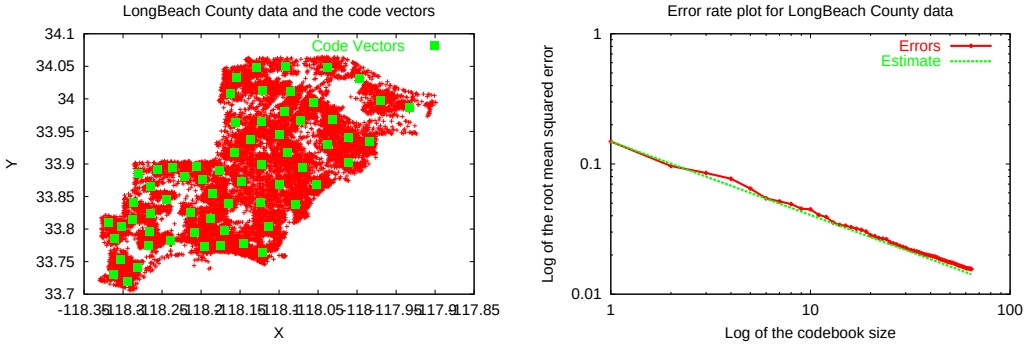


Figure 13: The Long Beach county data and the cluster centroids and the error rate plot. The error plot has slope -0.55142 and gives fractal dimension of 1.81350

performing the vector quantization which is computationally more intensive.

The computation of the correlation fractal dimension can be linear in the number of data points [27]. Thus the properties of a good vector quantizer can be easily determined from the estimate of the fractal dimension of the data set. We can now use this as a guideline for comparing and evaluating the performance of any vector quantization method.

The constant in our result is related to the error obtained when we use only one codeword. This error is proportional to the variance of the data set and is independent of the vector quantizer used. For a good vector quantizer, we expect the error rate to be related to the fractal dimension according to the relation in our result.

What does this mean?

Data Set	Time for VQ (sec)	Time for Corr Int (sec)	Fractal Dimension	-1/Slope
Sierpinski Triangle	24.64	1.207	1.60	1.65
Koch Curve	40.58	1.924	1.28	1.33
Diagonal Line	3.20	0.069	0.97	0.99
Montgomery County	207.74	12.398	1.80	1.84
Long Beach County	92.93	7.104	1.77	1.81

Table 1: Computational time for a codebook size of 64 and estimate of fractal dimension using correlation integral

- The performance of a Vector Quantization algorithm depends on the data set.
- We have a formula relating the performance of Vector Quantization and Fractal Dimension.
- We can use this relation to determine the optimal performance of any Vector Quantizer.
- The fractal dimension of a data set can be computed in linear time and can be used to determine the performance of a vector quantizer.
- This gives us a method to compare and evaluate the performance of a given Vector Quantization method.

6 Discussion and Contributions

The fractal dimension of the data set is a good indicator of the spread of the data. Thus, this can be used as an indicator for the amount of information hidden in the data. The computation of "fractal" dimension of a data set can be linear in the number of data points [27]. The fractal dimension estimates the intrinsic dimension of the data and hence the degrees of freedom of the data. Thus, this is a good indicator of the distribution of the data.

The *MaxFD* conjecture can be used to estimate the target dimension for a dimensionality reduction method. Also, we showed how we can use the result to compare the performance of different methods without having to apply different data mining methods to determine the amount of information retained. For a fixed dimension of the target space, we can use this result to compare and determine which dimensionality reduction methods performed better than the others.

The relation between the performance of a vector quantizer and the fractal dimension gives us an idea of the relation between the two concepts. This shows that the fractal dimension of a data set is closely related to the spread of the data and hence the information hidden in the data.

For each of the above tasks, we showed that we can use the fractal dimension of the data set can be used to make faster inferences about the data mining methods. The computation of the fractal dimension scales linearly in the number of data points. Thus, we can use this for evaluating the different methods quickly.

7 Conclusions

The "fractal dimension" of the data set is a good measure of the data distribution. It can be seen as a characterization of the spread of the data. It can be computed quickly and can be used to aid in different data mining tasks. Using the fractal dimension, we can make faster inferences about the performance of a method for a data mining task. We can also use this for evaluation and comparing different methods for their performance. We can do this in a way that is scalable to large data sets and avoid the expensive computations involved in implementing the various data mining tasks.

References

[1] R. E. Bellman. *Adaptive Control Processes*. Princeton University Press, Princeton, NJ, 1961.

[2] T. Berger. *Rate Distortion Theory*. Prentice-Hall, Englewood Cliffs, NJ, 1971.

[3] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, England, 1995.

[4] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[5] E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroqun. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

[6] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[7] D. DeMers and G. Cottrell. Non–linear dimensionality reduction. *Advances in Neural Information Processing Systems*, 5:580–587, 1993.

[8] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley Sons, 1973.

[9] C. Faloutsos and I. Kamel. Beyond uniformity and independence: Analysis of R- trees using the concept of fractal dimension. In *Proc. ACM SIGACT-SIGMOD-SIGART PODS*, pages 4–13, 1994.

[10] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. New York: Academic Press, 1990.

[11] R. G. Gallager. *Information Theory and Reliable Communication*. Wiley, New York, 1968.

[12] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.

[13] Z. Ghahramani and G. Hinton. The EM algorithm for mixtures of factor analyzers. Technical Report CRG-TR-96-1, University of Toronto, 1997.

[14] R. M. Gray. *Source Coding Theory*. Kluwer Academic Press, Boston, 1990.

[15] H. Abut, editor. Vector quantization. In *IEEE Reprint Collection*. IEEE Press, Piscataway, NJ, USA, 1990.

[16] I. T. Joliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[17] F. Korn. Ratio rules: A new paradigm for fast, quantifiable data mining. In *Proc. of the 24th International Conference on Very Large Data Bases(VLDB)*, pages 582–593, August 1998.

[18] Y. Linde, A. Buzo, and R. M. Gray. An Algorithm for Vector Quantizer Design. *IEEE Transactions on communications*, COM-28(1):84–95, Jan. 1980.

[19] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, IT-28(2):129–136, Mar. 1982.

[20] R. F. Murphy, M. V. Boland, and M. Velliste. Towards a systematics for protein subcellular location: Quantitative description of protein localization patterns and automated analysis of fluorescence microscope images. In *Proc Int Conf Intell Syst Mol Biol (ISMB 2000)*, pages 251–259, 2000.

[21] N. M. Nasrabadi and R. King. Image Coding Using Vector Quantization: A Review. *IEEE Transactions on Communications*, 36(8):957–971, Aug. 1988.

[22] R. T. Ng, H. V. Jagadish, and J. Madar. Semantic compression and pattern extraction with fascicles. In *Proc Int Conf on Very Large DataBases*, pages 186–197, September 1999.

[23] P. F. Swaszek, editor. Quantization. In *Benchmark Papers in Electrical Engineering and Computer Science*, volume 29. Van Nostrand Reinhold Company, Inc., New York, NY, USA, 1985.

[24] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1992.

[25] M. Schroeder and F. Chaos. *Fractal, Chaos and Power Laws*. Freeman, New York, 1991.

[26] D. Scott and J. Thompson. Probability density estimation in higher dimensions. In *Computer Science and Statistics: Proc. of the XV Symposium on the Interface*, pages 173–179, 1983.

[27] C. Traina, A. Traina, L. Wu, and C. Faloutsos. Fast feature selection using the fractal dimension. In *Proc. of XV Brazilian Symposium on Databases*, 2000.