HILBERT SPACE FILLING CURVE (HSFC) NEAREST NEIGHBOR CLASSIFIER

by

John David Reeder II

A thesis submitted in partial fulfillment of the requirements
for the Honors in the Major Program in Computer Engineering
in the College of Engineering and Computer Science
and in The Burnett Honors College
at the University of Central Florida
Orlando, Florida

ABSTRACT

The Nearest Neighbor algorithm is one of the simplest and oldest classification techniques. A given collection of historic data (Training Data) of known classification is stored in memory. Then based on the stored knowledge the classification of an unknown data (Test Data) is predicted by finding the classification of the nearest neighbor. For example, if an instance from the test set is presented to the nearest neighbor classifier, its nearest neighbor, in terms of some distance metric, in the training set is found. Then its classification is predicted to be the classification of the nearest neighbor. This classifier is known as the 1-NN (one-nearest-neighbor). An extension to this classifier is the k-NN classifier. It follows the same principle as the 1-NN classifier with the addition of finding k ($k > 1$) neighbors and taking the classification represented by the highest number of its neighbors.

It is easy to see that the implementation of the nearest neighbor classifier is effortless, simply store the training data and their classifications. The drawback of this classifier is found when a test instance is presented to be classified. The distance from the test pattern to every point in the training set must be found. The required computations to find these distances are proportional to the number of training points (N), which is computationally complex, especially with N large.

The purpose of this thesis is to reduce the computational complexity of the testing phase of the nearest neighbor by using the Hilbert Space Filling Curve (HSFC). The HSFC NN classifier was implemented and its accuracy and computational complexity is

compared to the original NN classifier to test the validity of using the HSFC in classification.

ACKNOWLEDGEMENTS

First and foremost I would like to thank God above for giving me the opportunity to attend the University of Central Florida, and for helping me achieve all that I have set out to accomplish

Second, I would like to thank my chair Dr. Georgiopoulos for guiding me through this process and allowing me to work with his group in Machine Learning. I look forward to starting graduate school with him as my advisor. Also, I would like to recognize my committee Dr. Bauer, and Dr. Gonzalez, for there comments and guidance in finalizing my paper

I would like to thank my family, for always believing in me and giving me to confidence to move away from home to pursue my college education. They have always supported me in every endeavor and without them none of this would be possible.

My friends Anton Kiriwas and James Ginn also deserve recognition for their help in reviewing my paper, and giving advice from their lessons learned on their own thesis’s.

Last but not least, I would like to thank Keisha Enfinger for her love and support through out my college career. With out her I would not have been able to succeed away from home.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVATIONS

HSFC	Hilbert Space Filling Curve
NN	Nearest Neighbor
PNN	Probabilistic Neural Network
N	Number of Training Points

Symbols used in Standard NN

x	A vector representing the input attributes. This vector is of dimensionality *n*.
k	Number of Neighbors used in *k*-NN
w	Weight used in Weighted *k*-NN

Symbols used in HSFC NN

n	Number of dimensions of Data
m	Order of HSFC
M	The number of bits in the HSFC index (*n*m*)
r	HSFC index
Byte	a word containing *n* bits
a_j	Vector $< a_1 a_2 ... a_n >$ representing a datum whose derived key is r .

Gaussian Database Naming Convention

gXc_YY	X represents Number of Classes YY represents % overlap
g2c_05	Gaussian 2-class 5% overlap

Chapter 1: Introduction

Classification is the problem of correctly classifying (labeling) data based on the features that they possess. For instance, an example of a classification problem is to recognize the type of fruit that we are dealing with (e.g., banana, apple, pear, etc.) based on certain features, such as color, smell, shape, and so on. In building such a classifier system one relies on domain knowledge about the problem at hand (e.g., that the color of the banana is yellow) and quite often on examples of data whose classification is known. The task then becomes to efficiently use this knowledge to design a classifier that performs this recognition (labeling), fast and accurately.

The Nearest Neighbor classifier is one of the oldest classifiers in use today. It has remained popular because of its simple implementation and its guaranteed error rate of less than twice the error rate of the Bayesian classifier, the best possible classifier when statistical information about the data is known (Cover and Hart, 1967). The nearest neighbor classifier can be described very simply. The classification (label) of a datum (of certain features) and unknown classification is the same as the classification of any datum of known classification whose features are closest in distance to the features of datum of unknown classification. The assumption here is that we have available to us data of known classification and the features of these data belong to a feature space for which meaningful distances can be defined.

One of the problems of the nearest neighbor classifier is the time that it takes to find the label of a datum of unknown classification. Assuming that we have stored the information about the features and labels of N known points to determine the classification of a new datum it requires that we calculate the distances of the new datum from all the stored points. This is a calculation that is proportional to N and it is prohibitively slow for large N. One of the ways that have been suggested in the literature to deal with this problem is the design of a prototype nearest neighbor. A prototype nearest neighbor is a nearest neighbor approach where compression of the stored data of known classification is performed first, by clustering. Then, when a new datum of unknown classification arrives its nearest prototype is first found and its classification is predicted to be identical with the classification of the prototype. Since the number of prototypes is, quite often, significantly smaller than the number of points that they represent this approach significantly reduces the computational complexity of the nearest neighbor approach. The disadvantage of the prototype nearest neighbor is that it results, at times, in the reduction of accuracy attained by the original nearest neighbor approach.

Examples of fast approaches that have been introduced into the literature to speed up the complexity of nearest neighbor include (Friedman, et al., 1975, Hart 1968). In particular, in Hart you start with a single randomly chosen observation as the training set, and then each additional data item is processed one at a time, adding it to the training set only if it is misclassified by the nearest neighbor rule computed on the current training set**.** While in Friedman the training data is stored in an optimized *k-d* tree, similar to a binary tree, allowing the algorithm to search only the training data that is sufficiently close to the test point for the nearest neighbor. The increase in speed comes from the decreased number of training points that are searched for the nearest neighbor.

Another way of dealing with the high computational complexity of the nearest neighbor classifier is by using the HSFC approach. Through this approach the N stored points of dimensionality n are mapped into a 1-dimensional index, called the HSFC index. The indexing happens in $O(N \log_2(N))$ time. Then when a new datum arrives the point with its closest index is found first (in $O(\log_2(N))$ time), and its classification is designated to be the classification of the point of closest index. What makes this approach successful, in addition to being computationally efficient, is that points whose Hilbert indexes are close are close in the original n – dimensional space. However, there might be points that are close in the original n – dimensional space whose Hilbert indices are not close. Hence, although Hilbert indexing improves the speed of determining a nearest neighbor, this advantage happens at the expense of not being able to find the nearest neighbor at all times.

This approach is similar to the method of (Skubalska-Rafajlowicz, et al, 1996) in the use of space filing curves to index the training data. In Skubalska-Rafajlowicz, the data was indexed using Peano and Sierspinski space filling curves. They show that these curves were able to produce classification performance on par with the standard NN approach. Their experiments were on very small data sets in the range of 80 to 200 Training points, and did not show the increase in classification speed. They dismissed the Hilbert space-filling curve saying it provided lower performance, but personal communication with Castro suggested that this was not the case. The experiments in this study focus on the Hilbert space-filling curve and its performance on data sets in the range of 2,000 to 500,000 training points, and dimensions ranging from 2 to 12.

The HSFC has been used in (Castro, 2004) to partition training data for the Fuzzy ARTMAP classifier. The training data was partitioned into separate sets and trained on individual ARTMAP classifiers. This approach decreased the training time for the large training set without negatively affecting the classification accuracy. This also allowed the problem to be implemented on a Beowulf cluster to further speed up the classification time.

The HSFC has also been used in (J.K.Lawder, et al., 200) to index multi-dimensional databases and to allow efficient querying on the indexed databases. Lawder's work provides step by step instructions for producing Hilbert indexes from multi-dimensional data that were helpful during the implementation of the HSFC NN classifier.

In the following sections several different nearest neighbor techniques are introduced and pseudo code for each is provided. Also an explanation of the HSFC and the procedure for indexing multi-dimensional data is provided with examples, followed by experiments comparing the performance of the HSFC NN classifier with the Standard NN classifier on several natural and generated databases. Experiments are also carried out on the very large Forest Cover database. Lastly avenues for future exploration are presented.

Chapter 2: Nearest Neighbor Algorithms

In order to validate the HSFC NN algorithm it is necessary to implement and test the standard NN algorithm. For the purposes of this thesis the 1-NN, k-NN and Weighted k-NN algorithms will be implemented, and will be subjected to the same testing process as the HSFC NN algorithm. In this way the Standard NN algorithms will act as a benchmark to gauge the success of the HSFC NN. In this section each of these algorithms are described and their classification process is detailed and analyzed.

2.1 1-NN Algorithm

The single nearest neighbor classification implemented by Cover and Hart is the most basic of all the NN implementations (Cover and Hart, 1967). Proposed in 1966, the paper showed that the single nearest neighbor implementation had a minimum probability of error that was less the twice the Bayes probability of error, and because the Bayes classifier is the optimum choice for a decision classifier, less than twice the probability of error for any classification rule. The single nearest neighbor implementation classifies the unknown observation by finding the nearest neighbor to the observation by some relevant distance function, and assigning its class to the unknown observation. Cover and Hart show that for a simple distribution the single nearest neighbor algorithm will have a probability of error that is less than the *k*-NN probability on the same distribution. This means that the 1-NN implementation is strictly better than the *k*-NN implementation for distributions where each in-class distance is larger than any of the between-class distances (Cover and Hart, 1967). The figure below is the set used in their example.

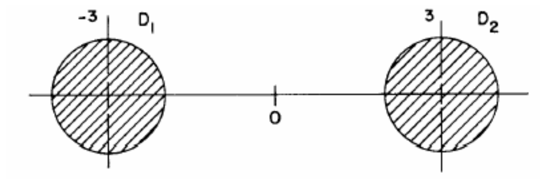

Figure 1: sample distribution used in example by Cover and Hart

2.1.1 1-NN Pseudo Code

Terminology:

$\mathbf{x}$: A vector representing the input attributes. This vector is of dimensionality *n*.

$\mathbf{x}_i^j$: The *i*-th vector of class *j*. The index *i* ranges from 1 to N_t^j, where N_t^j represents the number of vectors $\mathbf{x}$ that belong to class j. The index j ranges from 1 to *J*, where *J* represents the number of classes that the measured vector $\mathbf{x}$ could belong.

Step 1: We store all the points $\mathbf{x}_i^j$ in memory.

Step 2: We present the test pattern $\mathbf{x}$ to the 1-NN.

Step 3: We calculate the distance of $\mathbf{x}$ from all the stored patterns ($\mathbf{x}_i^j$'s). The distance of the pattern $\mathbf{x}$ from pattern $\mathbf{x}_i^j$ is defined as:

$$dis(x,\ x_i^j) = \sqrt{\sum_{l=1}^{d}(x(l) - x_i^j(l))^2}$$

Step 4: We find the minimum such distance. That is we find

$$dis(x,\ j_{\min}) = \min_{1\le j\le J,\ 1\le i\le N_t^j}\ dis(x,\ x_i^j)$$

Step 5: The predicted class for test pattern $\mathbf{x}$ is then class $j_{\min}$.

This process is repeated for every test pattern **x**, whose classification we want 1-NN to predict.

2.2 k-NN Algorithm

One of the short comings of the 1-NN approach is that it does not handle noise data very well. Noise data are points in the training set that are misclassified. If the test point's nearest neighbor is misclassified, then the algorithm will misclassify the test point. The *k*-NN algorithm overcomes this short-coming by considering the classification of multiple neighbors. During the performance phase the algorithm will select *k* neighbors, where *k* is a positive integer, and will give the test point the classification of the class with the highest number of representatives in the set of neighbors. Using this technique if one of the neighbors is misclassified the other neighbors will still give the correct classification. This also increases performance on databases that are not completely separable, when the data for different classes overlap each other. In these cases the decision boundary is blurred. A test point that is near a class boundary could have a neighbor that is correctly classified but still in the other class. Once again in this situation the 1-NN approach could misclassify the point, but the k-NN, choosing multiple neighbors, will classify the point correctly. In the following diagram an example of 5-NN is shown. In this example if the 1-NN rule is used the algorithm would choose class 2, but as you can see most of the points around the test point are class 1. The 5-NN rule shown will classify the test point as class 2, because 4 out of 5 of the neighbors selected are class 1.

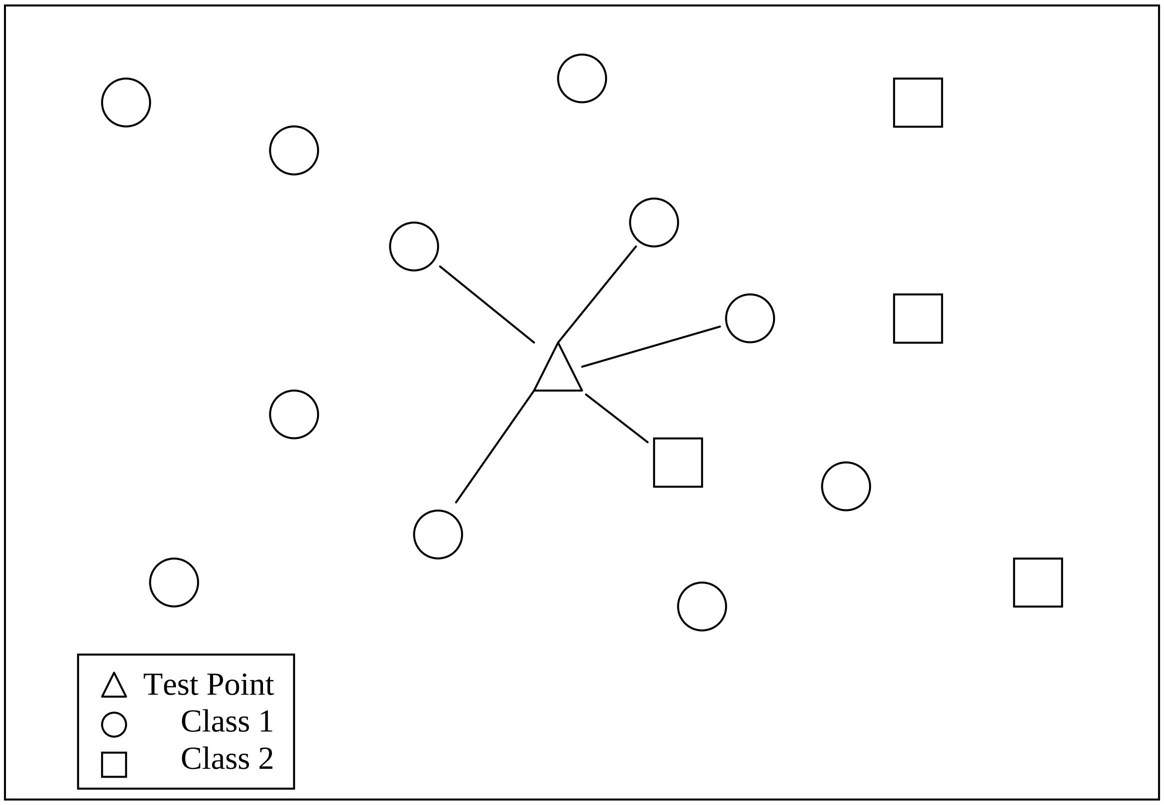

Figure 2: Example of 5-NN classification.

2.2.1 k-NN Pseudo Code

Terminology:

$\mathbf{x}$: A vector representing the input attributes. This vector is of dimensionality *n*.

$\mathbf{x}_i^j$: The *i*-th vector of class *j*. The index *i* ranges from 1 to N_t^j, where N_t^j represents the number of vectors $\mathbf{x}$ that belong to class j. The index j ranges from 1 to *J*, where *J* represents the number of classes that the measured vector $\mathbf{x}$ could belong.

Step 1: We store all the points $\mathbf{x}_i^j$ in memory.

Step 2: We present the test pattern $\mathbf{x}$ to the k-NN.

Step 3: We calculate the distance of $\mathbf{x}$ from all the stored patterns ($\mathbf{x}_i^j$'s). The distance of the pattern $\mathbf{x}$ from pattern $\mathbf{x}_i^j$ is defined as:

$$dis(x, x_i^j) = \sqrt{\sum_{l=1}^{d} (x(l) - x_i^j(l))^2}$$

Step 4: We find the *k* minimum such distances. We call the class labels corresponding to these *k* smallest such distances as:

$$j_{\min}^1,\ j_{\min}^2,\ \ldots,\ j_{\min}^k$$

Step 5: The predicted class for test pattern $\mathbf{x}$ is class $j_{\min}$. Class $j_{\min}$ is the class that appears more often in the discrete set $\{j_{\min}^1, j_{\min}^2, \ldots, j_{\min}^k\}$.

This process is repeated for every test pattern $\mathbf{x}$, whose classification we want k-NN to predict.

2.3 Weighted k-NN algorithm

The Weighted k-NN algorithm is almost identical to the standard k-NN algorithm with the exception that each neighbor is given a weight depending on its distance from the test point. The classifier decision is then based on the weights. The class with the highest weight is the classification chosen for the test point. In most situations this method will choose the same classification as the k-NN. The difference between the two becomes useful when the test data is sparse. In a sparse dataset the training points are spread thin throughout the training space, this means that during the k-NN classification the algorithm might have to look far away from the test point to find *k* neighbors. It is possible that in looking for neighbors the algorithm will cross the decision boundary and select points of the wrong classification. The Weighted k-NN method is better suited for this situation because it computes a weight for each neighbor based on the distance from the test point. Neighbors that are very close to the test point receive a higher weight than the neighbors that are further away. The weights of all of the neighbors with the same classification are summed and the class with the highest weight is chosen for the test point. The figure below shows an example of this situation. The k-NN classifier would choose class 1 because the 3 out of 5 neighbors are class 1, but the weighted k-NN would choose class 2 because the two neighbors of class 2 are much closer to the test point than the 3 neighbors of class 1.

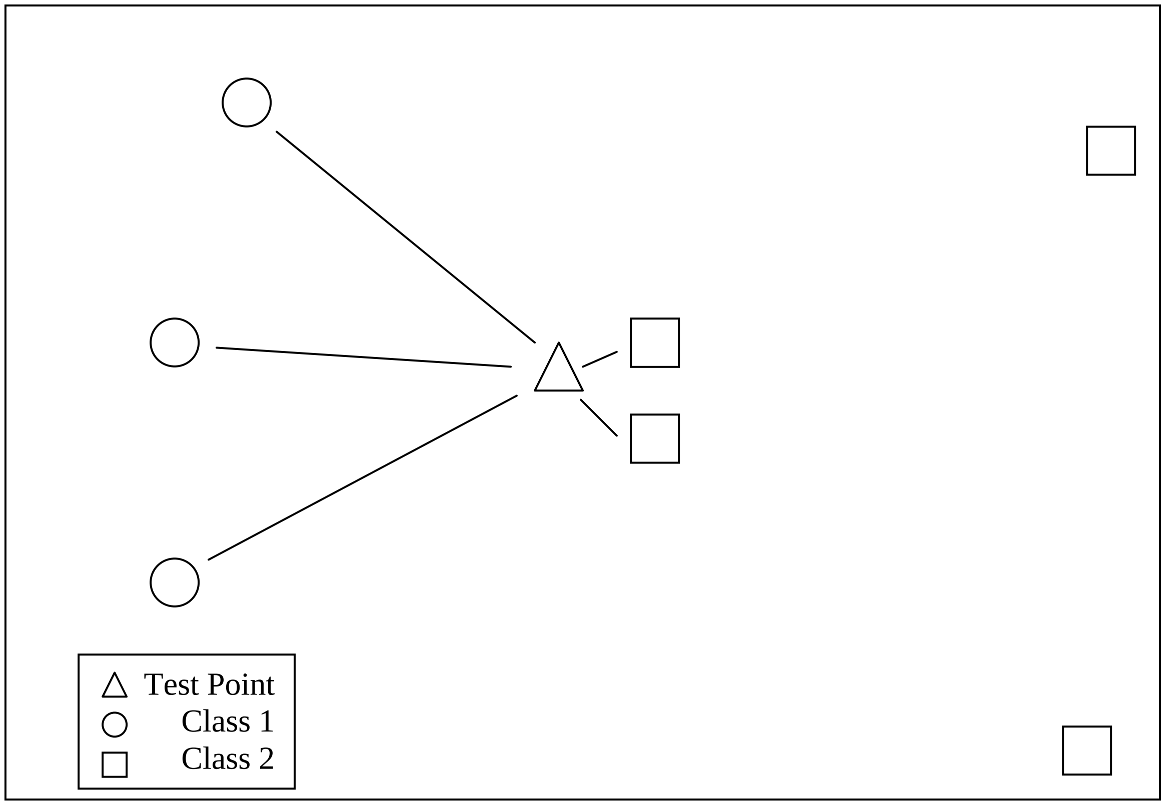

Figure 3: Example of Weighted 5-NN

2.3.1 Weighted k-NN Pseudo Code

Terminology:

$\mathbf{x}$: A vector representing the input attributes. This vector is of dimensionality *n*.

$\mathbf{x}_i^j$: The *i*-th vector of class *j*. The index *i* ranges from 1 to N_t^j, where N_t^j represents the number of vectors $\mathbf{x}$ that belong to class j. The index j ranges from 1 to *J*, where *J* represents the number of classes that the measured vector $\mathbf{x}$ could belong.

w : The weight of a training point in relation to the test point. The closer the training point is to the test point the higher the weight.

Step 1: We store all the points $\mathbf{x}_i^j$ in memory.

Step 2: We present the test pattern $\mathbf{x}$ to the k-NN.

Step 3: We calculate the distance of $\mathbf{x}$ from all the stored patterns ($\mathbf{x}_i^j$'s). The distance of the pattern $\mathbf{x}$ from pattern $\mathbf{x}_i^j$ is defined as:

$$dis(x, x_i^j) = \sqrt{\sum_{l=1}^{d} (x(l) - x_i^j(l))^2}$$

Step 4: We find the *k* minimum such distances. We call the class labels corresponding to these *k* smallest such distances as:

$$j_{\min}^1,\ j_{\min}^2,\ \ldots,\ j_{\min}^k$$

Step 5: We find the weighted distance of the presented pattern $\mathbf{x}$ from each of the k closest points (found in Step 4). That is, we calculate (for every l, such that $1 \le l \le k$)

$$dis_n(\mathbf{x}, \mathbf{x}_{(\cdot)}^{j_{\min}^l}) = \frac{dis(\mathbf{x}, \mathbf{x}_{(\cdot)}^{j_{\min}^l})}{\sum_{r=1}^{k} dis(\mathbf{x}, \mathbf{x}_{(\cdot)}^{j_{\min}^r})}$$

$$w = 1 - \frac{dis(x, x_i^j)}{\sum_{i=0}^{k} dis(x, x_i^j)}$$

Step 6: Weighted distances, calculated in Step 5, that are calculated from points belonging to the same class are added together. The predicted class for test pattern $\mathbf{x}$ is class $j_{\min}$, which is the class from the group of classes $\{ j_{\min}^1, j_{\min}^2, ..., j_{\min}^k \}$ that produces the largest such sum of weighted distances.

This process is repeated for every test pattern $\mathbf{x}$, whose classification we want k-NN to predict.

Chapter 3: The Hilbert Space Filling Curve

A Space-filling curve is a continuous map of a one-dimensional interval) into a two-dimensional area (a plane-filling function) or a three-dimensional volume.

The Hilbert curve is used to map an n-dimensional coordinate system to a 1-dimensional index. The Hilbert curve is known to maintain some of the spatial relationships of the n-dimensional space. This trait makes is useful in clustering multi-dimensional data.

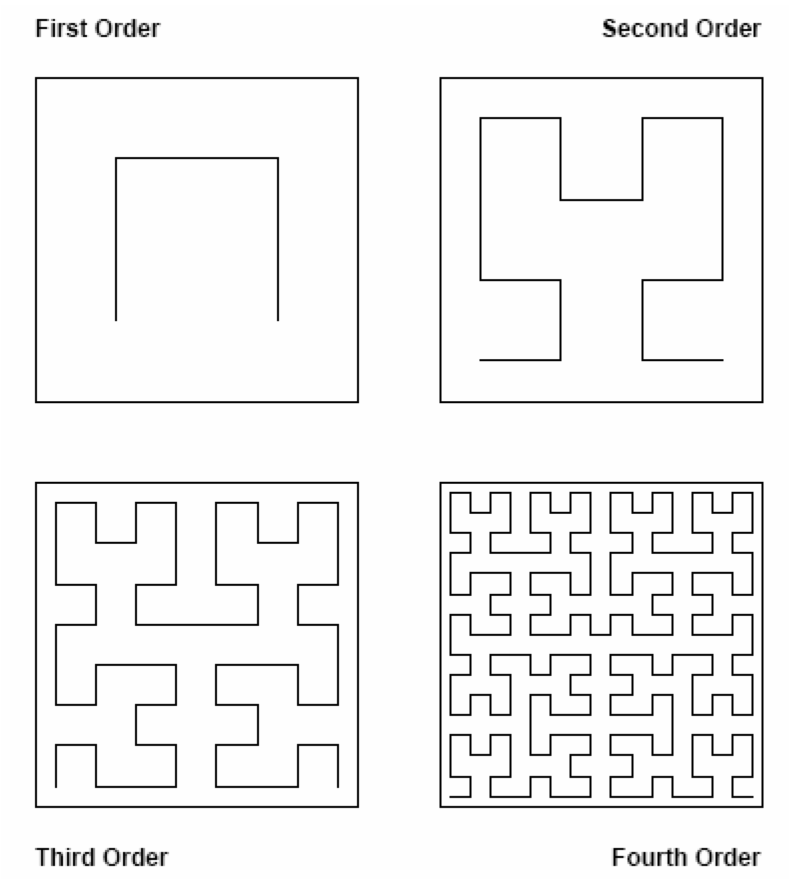

Figure 4: 2-D Examples of Hilbert Space filling curves

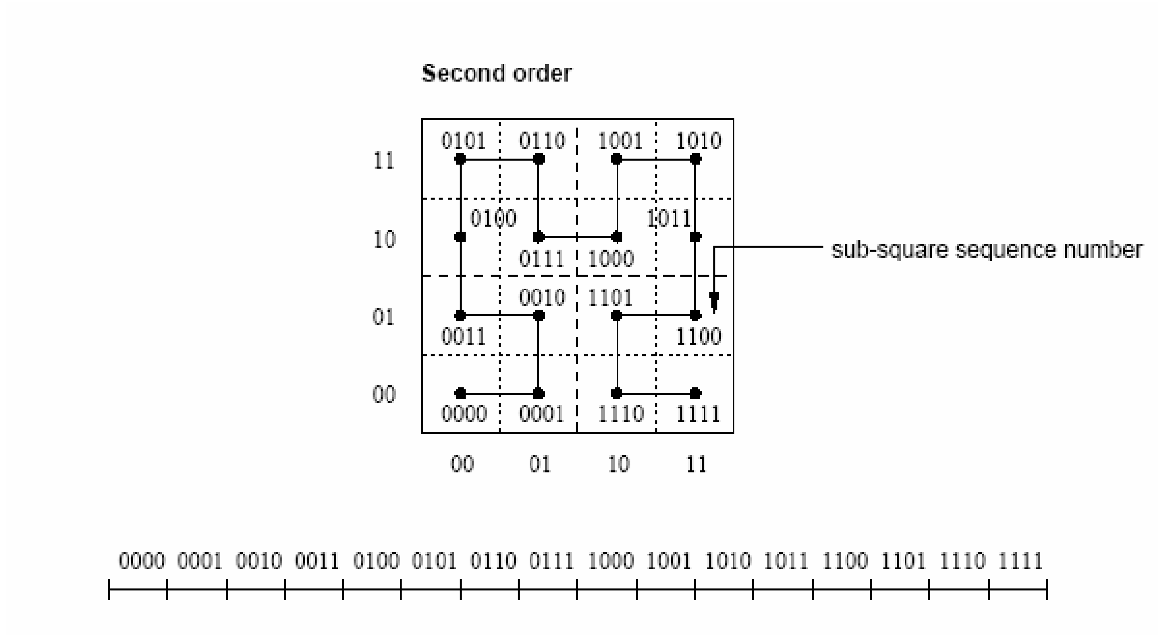

Figure 5: Mapping 2-D coordinates to 1-D Hilbert Index

The HSFC converts an *n*-dimensional space to a 1-dimensional space. In the implementation of the HSFC NN algorithm this property is used to convert the Training Data to a 1-dimensional array. This array is then sorted on the Hilbert index making it possible to search the training space in $O(N \log_2(N))$ time using a quick sort or a merge sort. The Hilbert Index is found by using Butz's Algorithm (Butz, 1971) to convert the *n*-dimensional coordinates to a Hilbert Index.

3.1 Butz's Algorithm

This section provides the pseudo code for Butz's Algorithm, and gives simple examples of the mapping process from 2-D coordinates to the 1-D Hilbert index, and the reverse operation.

Algorithm Definitions

n : number of dimensions

m : the order of approximation

M : the number of bits in a *derived–key* (*n*m*)

r : an *N*–Bit binary Hilbert *derived–key* expressed as a real number in the range [0, 1).

byte : a word containing *n* bits

ρ_j^i : where $i \in \{1,...,n\}$ and $j \in \{1,...,m\}$ A binary digit in r such that:

$$r = 0.\rho_1^1\rho_2^1...\rho_n^1\rho_1^2\rho_2^2...\rho_n^2\rho_1^3\rho_2^3...\rho_n^m$$

ρ^i : i^{th} binary byte in r . $\rho^1 = \rho_1^1\rho_2^1...\rho_n^1$

a_j : A coordinate in dimension j of the point $< a_1a_2...a_n >$ whose derived key is r . It is also expressed as a real number between [0, 1).

α_j^i : A binary digit in the coordinate a_j such that $a_j = \alpha_j^1\alpha_j^2...\alpha_j^m$

Principal position **J:** The first bit from the right that is different. If all of the bits are the same then it is the least significant bit (or the one farthest to the right). Position is counted from the left.

Ex.
00110011 J = 6
01000000 J = 2
11111111 J = 8

Parity: Even or Odd depending on the number of 1's in a byte.

Ex.
01100110 = Even
01010001 = Odd

2-D Example: Hilbert Index to 2-D Coordinate

Variables used in conversion:

1. J_i : Principal position of ρ^i

2. σ^i : Grey code of ρ^i . $\sigma_1^i = \rho_1^i, \sigma_2^i = \rho_2^i \oplus \rho_1^i, ... \sigma_n^i = \rho_n^i \oplus \rho_{n-1}^i$ where σ_n^i and ρ_n^i are the n^{th} bits of σ^i and ρ^i .

3. τ^i : Obtained by complimenting σ^i in the n^{th} position and if it is odd parity then complimenting it in the Principal Position.

4. $\tilde{\sigma}^i$:Circular Shift σ^i to the *right* by $\sum_{k=1}^{i-1} J_k$

5. $\tilde{\tau}^i$:Calculated the same way as $\tilde{\sigma}^i$: using τ^i

6. $\omega^i := \omega^{i-1} \oplus \tilde{\tau}^{i-1}$, where $\omega^1 \equiv (0,0,0,...0)$

7. α^i : $\omega^i \oplus \tilde{\sigma}^i$

2-D Example:

$n = 2$
$m = 2$
$M = 4$
ρ^1 = First 2 bits of r
ρ^2 = Second 2 bits of r

This algorithm is an iterative algorithm. There are m iterations. In the following steps i represents then iteration number.

Let $r = 0.0010$
$\therefore \rho^1 = 00$, $\rho^2 = 10$

When converting from a Hilbert Index to an n-dimensional coordinate we want to find the value of α^i from r . Also most of the calculations for multiple iterations can be done simultaneously.

1. **Find** J_i

   $J_1 = 2$ $J_2 = 1$

2. **Calculate** σ^i

   The first bit of σ^i matches the first bit of ρ^i. The second bit of σ^i is the first and second bits of ρ^i XORed with each other.

   $\sigma^1 = 00$ $\sigma^2 = 11$

3. **Calculate** τ^i

   Compliment σ^i in the n^{th} position and if odd parity compliment the principal position.

   $\tau^1 = 00$ $\tau^2 = 00$

4. **Calculate** $\tilde{\sigma}^i$

   Never shift the first iteration. The second iteration and after shift right

   $$\sum_{k=1}^{i-1} J_k$$

   $\tilde{\sigma}^1 = 00$ $\tilde{\sigma}^2 = 11$

5. **Calculate** $\tilde{\tau}^i$

   Calculate the same way as $\tilde{\sigma}^i$.

   $\tilde{\tau}^1 = 00$ $\tilde{\tau}^2 = 00$

6. **Calculate** ω^i

   $\omega^1 = 00$ otherwise $\omega^i = \omega^{i-1} \oplus \tilde{\tau}^{i-1}$

   $\omega^1 = 00$ $\omega^2 = 00$

7. **Calculate** α^i

   $\alpha^i = \omega^i \oplus \tilde{\sigma}^i$

   $\alpha^1 = 00$ $\alpha^2 = 11$

8. **Convert** α^i **to** a_j

   α^i is a vector containing the i^{th} bit of each coordinate. Therefore α^1 contains the first bits of each coordinate a_j and α^2 contains the second bit of each coordinate a_j.

   $a_1 = 0.01$ $a_2 = 0.01$

2-D Example: 2-D coordinates to Hilbert Index

Variables used in conversion:

1. $\omega^i = \omega^{i-1} \oplus \tilde{\tau}^{i-1}$, where $\omega^1 \equiv (0,0,0,...0)$
2. $\tilde{\sigma}^i : \alpha^i \oplus \omega^i$, where $\tilde{\sigma}^1 \equiv \alpha^1$
3. σ^i : circular shift $\tilde{\sigma}^i$ *left* $\sum_{k=1}^{i-1} J_k$ times. First iteration does not shift.
4. ρ^i : $\rho_1^i = \sigma_1^i, \rho_2^i = \rho_1^i \oplus \sigma_2^i, ..., \rho_n^i = \rho_{n-1}^i \oplus \sigma_n^i$
5. J_i : principal position of ρ^i
6. τ^i : Obtained by complimenting σ^i in the n^{th} position and if it is odd parity then complimenting it in the Principal Position.
7. $\tilde{\tau}^i$:Calculated the same way as $\tilde{\sigma}^i$: using τ^i

When calculating the Hilbert Index from a coordinate each value must be calculated in the order above; one iteration at a time. The steps below have the values for each iteration side by side, but each iteration must be completed before continuing to the next.

2-D Example:

$n = 2$
$m = 2$
$M = 4$

$a_1 = 0.01$	$a_2 = 0.01$
$\alpha^1 = 00$	$\alpha^2 = 11$

We want to find ρ^i, we already know α^i

1. **Calculate** ω^i
   ω^1 is always 00, each other iteration is $\omega^{i-1} \oplus \tilde{\tau}^{i-1}$
   $\omega^1 = 00$ $\qquad\qquad$ $\omega^2 = 00$

2. **Calculate** $\tilde{\sigma}^i$
   $\tilde{\sigma}^1 \equiv \alpha^1$ each other iteration is $\alpha^i \oplus \omega^i$
   $\tilde{\sigma}^1 = 00$ $\qquad\qquad$ $\tilde{\sigma}^2 = 11$

3. **Calculate** σ^i
   Rotate $\tilde{\sigma}^i$ circular left according to rule above. First iteration never rotates.
   $\sigma^1 = 00$ $\qquad\qquad$ $\sigma^2 = 11$

4. **Calculate** ρ^i

   $\rho_1^i = \sigma_1^i, \rho_2^i = \rho_1^i \oplus \sigma_2^i, \ldots, \rho_n^i = \rho_{n-1}^i \oplus \sigma_n^i$

   $\rho^1 = 00$ $\rho^2 = 10$

5. **Calculate** J_i

   principal position of ρ^i

   $J_1 = 2$ $J_2 = 1$

6. **Calculate** τ^i

   Obtained by complimenting σ^i in the n^{th} position and if it is odd parity then complimenting it in the Principal Position.

   $\tau^i = 00$ $\tau^i = 00$

7. **Calculate** $\tilde{\tau}^i$

   Calculated the same way as $\tilde{\sigma}^i$: using τ^i

   $\tilde{\tau}^i = 00$ $\tilde{\tau}^i = 00$

8. **Find** r **from** ρ^i

   After completing all iterations the Hilbert index $r = 0.\rho^1\rho^2 \ldots \rho^m$

   $r = 0.0010$

The following table and figure show examples of the Iris Plant Database mapped to the Hilbert Space filling curve. As you can see from the Figure the HSFC does a good job of clustering the data. The clustering properties of the Hilbert Space Filling curve are what make it possible to implement the NN algorithm based on the Hilbert index. As shown in the figure below the classes are easily separated, leading to very high accuracy in the classification phase.

Table 1: Examples of HSFC Mapping

Class	Attr 1	Attr 2	Hilbert Index
Versicolor	0.450113	0.159633	0.194565
Virginica	0.653459	0.679114	0.534393

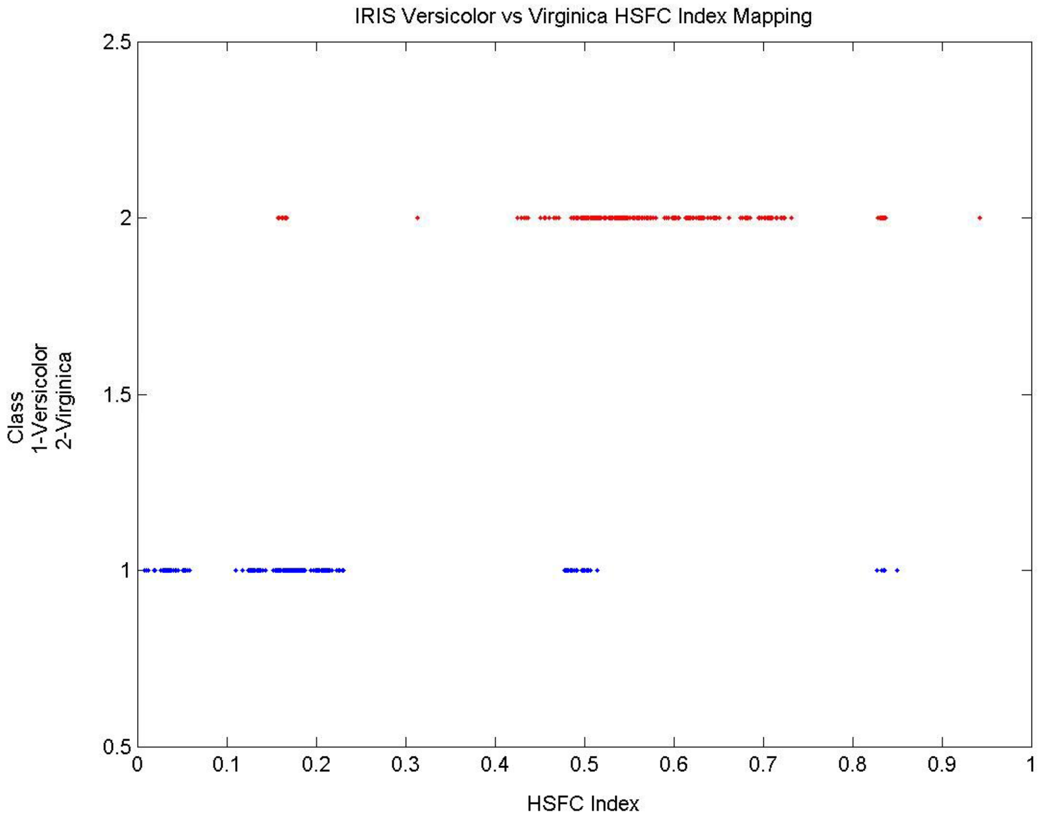

Figure 6: Iris Plant Database mapped to the HSFC

3.2 HSFC NN Pseudo Code

Terminology (Hilbert NN)

n : Number of dimensions
m : Order of approximation
M : The number of bits in a *derived key* $(n*m)$
r : An $N-bit$ binary Hilbert *derived key* where $r \in [0,1)$
a_j : An n-dimensional vector representing the input attributes

1-NN Hilbert

```
Step 1: Store Training Set in an Array

Step 2: For each instance in the Training set use Butz's Algorithm to convert $a_j$ to $r$

Step 3: Sort the Training Set by $r$

Step 4: Use Butz's algorithm to convert $a_j$ to $r$ for each instance to be classified

Step 5: Use a Binary Search based on $r$ to find the Training Instances before and after
        your test instance on the Hilbert Curve.

Step 6: Test the difference between both Training Instances $r$ and the Test Instances $r$
        Assign the Test Instance the class of the Training Instance that meets the
        following criteria
                        $\min(|r_{T1} - r_C|, |r_{T2} - r_C|)$
        Where $r_{T1}$ and $r_{T2}$ are the Hilbert Key's of the Training Instances and $r_C$ is the
        Hilbert Key of the Test Instance.
```

Chapter 4: Experimental Results

This section details the experiments that were run to compare the HSFC NN and the Standard NN. It gives a brief overview of the experimental procedure, as well as presenting the results of the experiments.

4.1 Experimental Procedure

The experiments were carried out in MATLAB using the HSFC_NN_Alg.dll. This DLL is capable of performing both the Standard NN and the HSFC NN. The datasets are stored in MAT files, and M-file scripts control the flow of the experiments. The *RunNNExperiments* script runs the experiment routine on each database, and the *TestHSFCNNAlg* script controls the flow of each individual experiment. The *TestHSFCNNAlg* script takes 3 arguments; DBName, NumClass, and Hilbert. DBName is the name of the database to be tested; this matches the beginning of the file name that contains the data, NumClass is the number of classes of the data, and Hilbert is a Boolean variable, true if you want to use the HSFC, or false if you want to use the Standard NN. The sequence of a single experiment is as follows:

1. Load the Training, Test, and XV sets from the mat files for the database.
2. Test Phase - Run the Algorithm in One NN mode, if using the HSFC the M parameter is set to 7.

3. Cross Validation for *k*-NN Unweighted – The algorithm is run repeatedly, each time changing the parameters *m* and *k* for HSFC and only *k* for Standard NN. The *m* parameter is ranged from 7:12 and the *k* parameter is ranged from 2:20.
4. Test Phase - the algorithm is run once in *k*-NN Unweighted mode using the parameters *m*max and *k*max – the parameters that maximized the performance in the cross validation phase.
5. Cross Validation for *k*-NN Weighted – The algorithm is run repeatedly, each time changing the parameters. M and K for HSFC and only K for Standard NN. The *m* parameter is ranged from 7:12 and the *k* parameter is ranged from 2:20.
6. Test Phase – The algorithm is run once in *k*-NN Weighted mode using the parameters *m*max and *k*max – The parameters that maximized the performance in the cross validation phase.
7. Output Results – The HSFC_NN_Alg returns the Accuracy, an array of the Result classification, and the running time of the algorithm. The *TestHSFCNNAlg* script creates a mat file that contains the accuracy, running times, and result class array for each of the test phase runs of the algorithm, it also stores the running time for the entire experiment. The mat file is named <DBName>_HSFC_Results.mat or <DBName>_NN_Results.mat depending on which mode the experiment was run in.

The *TestHSFCNNAlg* script is run on each of the fifteen databases. After this the *OutputResults* script is run to create a comma delimited text file that contains the accuracy and runtimes for both the HSFC NN and the Standard NN for each database.

4.2 Test Databases

This section will give a short description of each of the databases that were used in the experiments. Some of these databases are artificially generated while others were obtained from the UCI repository found here:
http://www.ics.uci.edu/~mlearn/MLRepository.html

4.2.1 Gaussian Databases

Twelve of the fifteen databases used in testing are artificially created Gaussian databases. These databases are created by setting a mean value for the class and filling in the points around the mean using a Gaussian distribution. All of these databases are dimensionality n = 2. The differences in each of the databases are the number of classes and the percent overlap of the classes. The number of classes available are 2, 4, and 6, and the percent overlap values are 5 %, 15 %, 25 %, and 40 %. The percent overlap determines the maximum accuracy that you can expect for each database. With a 5 % the maximum accuracy that you can expect will be 95 %. This is because the 5 % of the data that are overlapping have an equal probability to be any of the classes.

4.2.2 IRIS Plant Database

The Iris Plant Database was donated to the UCI Repository by Michael Marshall, and was created by R.A. Fisher in 1936. It contains 3 classes with 50 instances each. Each instance has four numeric attributes, and the purpose of the data is to classify the type of iris plant. The four attributes are the sepal length in cm, the sepal width in cm, the petal length in cm, and the petal width in cm. The classes are iris setosa, iris versicolour, and iris virginica. The underlying statistics for this data set are available, as well as any other documentation needed. This data base is one of the most widely used because it has been around for a long time, and it is very well documented. One of the drawbacks of this database is that it has very few instances. The IRIS database used in this research has been modified by adding points around the original data points increasing the total number of observations. Also the data has been reduced to a dimensionality of n = 2 by choosing two dimensions where the classes are linearly separable. And lastly one of the classes has been dropped from the database. The final database is a 2 dimensional, 2 class problem with around 5 % overlap in the classes.

4.2.3 Abalone Database

The abalone database was donated to the UCI repository in 1995 by Sam Waugh. The classification task is to predict the age of an abalone from physical measurements. The normal process for determining the age of an abalone is to cut the shell through the cone, stain it, and then count the number of rings. The task is to classify the age using easier to obtain measurements. This database has 29 classes and 8 attributes. This database has also been altered by generating instances from the original data.

4.3.4 Page blocks Database

The abalone database was donated to the UCI repository in 1995 by Donato Malerba. The problem consists of classifying all of the blocks of the page layout of a document that have been identified by a segmentation process. This is an important step in the analysis of a document. There are 5 classes; text (1), horizontal line (2), picture (3), vertical line (4), and graphic (5). The database has ten numeric attributes taken from measurements of each block, has very little noise, and has no missing data. This database has also been altered by generating new instances.

4.3 Results

The results are presented in the table below. The run times in the table below are measured in the HSFC_NN_Alg.cpp file. The number of clock cycles that pass while the algorithm is running are measured and then divided by the number of clock cycles per second. This time includes the Training Phase and the Performance phase of both algorithms.

Table 2: Results on small databases

Database Name	Type	One NN	Run Time	*k*-NN	Run Time	*k*-WNN	Run Time
g2c_05	HSFC	91.56	0.016	94.34	0.031	95.04	0.031
g2c_05	NN	92.04	1.281	95.14	1.375	95.12	1.359
g2c_15	HSFC	79.4	0.015	84.5	0.031	84.42	0.047
g2c_15	NN	78.52	1.281	84.22	1.375	84.22	1.375
g2c_25	HSFC	64.9	0.032	73.22	0.031	73.36	0.031
g2c_25	NN	67.2	1.281	73.84	1.375	73.84	1.375
g2c_40	HSFC	53.98	0.031	58.58	0.032	58.44	0.032
g2c_40	NN	52.68	1.281	57.98	1.391	57.98	1.375
g4c_05	HSFC	89.46	0.015	94.76	0.031	95.06	0.031
g4c_05	NN	92.2	1.265	94.74	1.407	94.56	1.39
g4c_15	HSFC	76.74	0.016	83.88	0.031	83.94	0.047
g4c_15	NN	78.38	1.281	83.98	1.407	83.68	1.406
g4c_25	HSFC	65.82	0.031	74.7	0.032	74.88	0.031
g4c_25	NN	65.86	1.297	74.74	1.359	74.5	1.344
g4c_40	HSFC	49.32	0.031	57.78	0.032	57.58	0.031
g4c_40	NN	48.02	1.266	58.44	1.375	58.08	1.375
g6c_05	HSFC	90.167866	0.031	91.08713	0.031	92.825739	0.031
g6c_05	NN	93.105516	1.281	94.664269	1.39	94.684253	1.375
g6c_15	HSFC	77.278177	0.016	80.695444	0.031	80.935252	0.032
g6c_15	NN	75.979217	1.281	84.572342	1.453	84.572342	1.438
g6c_25	HSFC	62.589928	0.032	69.744205	0.032	70.063949	0.032
g6c_25	NN	64.128697	1.343	72.941647	1.422	72.781775	1.484
g6c_40	HSFC	45.043965	0.031	54.236611	0.031	53.816946	0.031
g6c_40	NN	44.964029	1.312	56.714628	1.406	57.394085	1.39
Iris_DrG_Norm	HSFC	90.979167	0.016	94.333333	0.031	94.416667	0.031
Iris_DrG_Norm	NN	91.041667	1.218	93.833333	1.265	93.8125	1.328
new_abalone_500_Norm	HSFC	52.176279	0.015	52.339499	0.016	51.904244	0.016
new_abalone_500_Norm	NN	49.945593	1.297	52.774755	1.343	52.50272	1.344
pageblocks_Norm	HSFC	71.531966	0.032	86.529956	0.032	86.489747	0.031
pageblocks_Norm	NN	88.661037	2.484	90.349819	2.484	90.269401	2.5

In the table above the g2c_05 database the HSFC 1-NN has a running time of 0.016 seconds and the Standard NN has a running time of 1.281 seconds. This means that the HSFC 1-NN performs the Training Phase and classifies the entire test set in 0.016 seconds. The training phase of the standard NN is simply passing the Training Data array, so this means that it only has to classify the Test Set. It takes the Standard NN 1.281 seconds to classify the test set. This means that for the g2c_05 database the HSFC NN gives a 98.75% decrease in the time needed to classify the test set while maintaining nearly the same accuracy.

Table 2 shows that the HSFC NN algorithm maintains the accuracy of the NN algorithm in almost all of the databases, and increases the accuracy in a few instances. However, there are exceptions. In the Gaussian 6 class problems the HSFC algorithms were outperformed by the NN algorithms by 2 to 4 %, and on the page blocks database it was outperformed by 3 to 16%. These discrepancies are caused by the fact that the HSFC does not save all of the spatial relationships between data; it only preserves some of them. Because of this the HSFC will choose slightly different neighbors than the standard NN. In most of the databases choosing different neighbors still leads to the correct classification. However in the 6-class Gaussian database the decision boundaries cross over the quadrant boundaries. One of the properties of the HSFC is that it will naturally divide the data up into quadrants, the 2 and 4 class Gaussian databases are easily separated into quadrants and because of this the performance of the HSFC NN does not suffer. The 6-class problem however is not easily separated by quadrants and because of this the HSFC NN has a higher possibility of choosing the wrong neighbors. This and the higher number of dimensions are the reason for the discrepancies in the page blocks data.

One possible solution to this problem would be to use multiple curves, shifting each curve slightly so that neighbors that are close in *n* space will be close on one of the curves. After the three curves are created they would then vote on the classification of a test point. This technique should decrease the error associated with missing actual neighbors, and will be experimented with in the future.

During the cross validation phase of the experiments the algorithms were run on validation sets to determine the optimal values of *k* and *m*. This is done by running the algorithm multiple times each time changing the values of the parameters. The values that produce the best accuracy are selected as the optimal values and the experiments are run on the test set using these values. During the cross validation some observations were made about the effect of the parameters on the accuracy of the algorithms. The figures below show the values of *k* and *m* and there effect on the accuracy of the HSFC algorithm. Figure 7 shows that as *k* increases the accuracy increases up until around $k = 9$, after that the accuracy levels off and remains nearly the same for all greater values of *k*. For the order of the curve *m* it was found that the accuracy fluctuated until an order was reached that assigned each of the training points there own index. For lower orders of the Hilbert curve multiple training points would be mapped to the same index. The sorting algorithm used in the HSFC NN is a quick sort. This sorting algorithm is only partially determinant. This means that if two values have the same index then each time the algorithm is run their order in the HSFC could change. Once the order of the curve was high enough to assign each training point a unique index the accuracy stopped fluctuating. That order also produced the best accuracy, any increase in the order after this point did not affect the accuracy of the algorithm. From these observations it was determined that the best choice for the order of the curve, is the order that assigned each training point a unique index. The selection of the quick sort as the sorting algorithm was arbitrary, any $O(N \log_2(N))$ sorting algorithm would suffice. For example a merge sort, which is completely determinant, could also be used, and would resolve the fluctuating accuracy at lower orders.

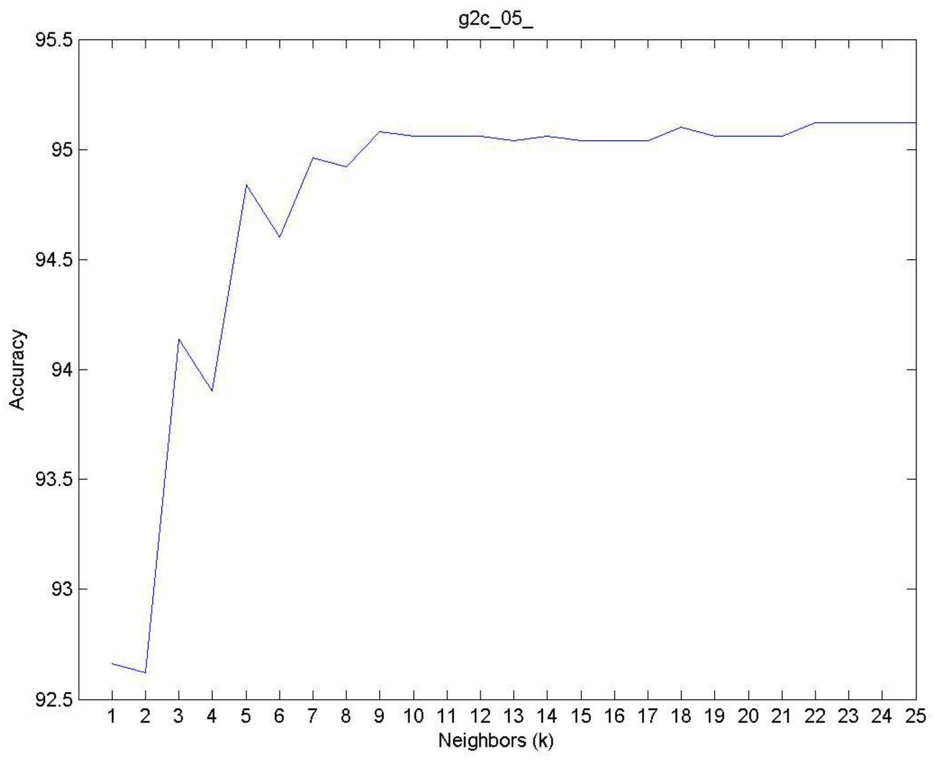

Figure 7: Gaussian 2 class Accuracy vs. K

A second set of experiments were run on the Gaussian 2-class 5 % database to show the effects of training set size on classification time. The experiments were run again using training set sizes of 500, 1000, 1500, and 2000. The points were obtained by converting some of the points in the cross validation set to training points. The experiments were run exactly the same as they were in the original experiments. The results are shown in the table and figure below.

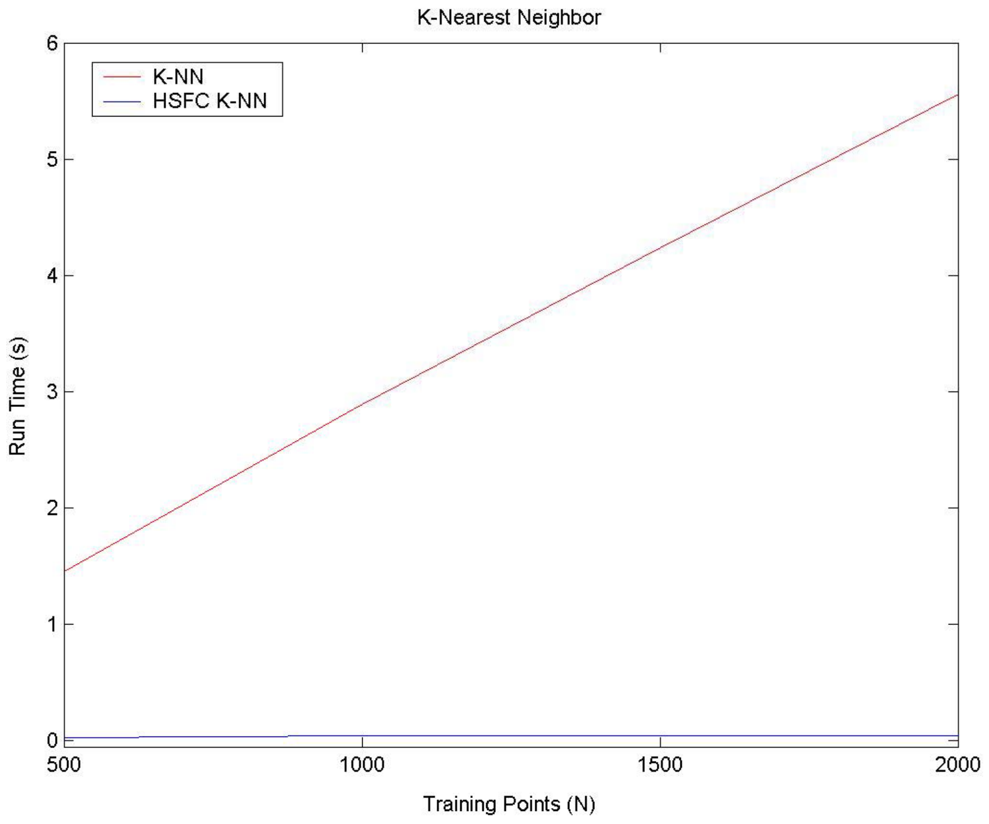

Figure 8: Run Time vs. Training set size

Figure 8 shows the run time vs. the Training Set size, as you can see the run time increases linearly with training set size for the standard k-NN algorithm but the HSFC NN run time is barely affected. This same experiment will be run again on a much larger database so that the training set size can have a much larger domain, and the differences in the run times of both algorithms will be more pronounced. The results from this second experiment show that the HSFC NN is significantly more efficient than the standard NN.

4.3.1 Large Database Experiments: Forest Cover Database

To further test the speed increase of the HSFC NN experiments were run on the Forest Cover Database. This database was obtained from the UCI KDD archive. It consists of 581,012 instances, and 7 classes. The classification task of this database is to determine the Forest Cover type from measurements acquired from a parcel of land. Of the 54 attributes, 10 of them are quantitative, and the other 44 are binary representations of 2 qualitative attributes. The HSFC has a hardware limitation, such that the number of dimensions times the order of the curve must be less than 64 ($n*m < 64$) on a 32-bit machine. In order to classify the Forest Cover Database the 44 binary attributes were converted to two binary strings, and from there to two decimal attributes. This reduced the number of dimensions to 12 allow the use of an order 5 curve. After this each attribute was normalized, and the database was split into classes. To test the scale up of both algorithms, the database was used to create data sets with training sizes ranging from 2k – 512k in increments of $2^i k$, where i ranges from 1 to 9. The algorithm was then run on each of these datasets. The results are presented in the following table.

Table 3: Forest Cover Results

DBName	Alg Type	One NN	Run Time	*k*-NN	Run Time	*k*-WNN	Run Time
ForestCover_2k_	HSFC_NN	45.426829	0.141	48.970412	0.172	48.65054	0.188
ForestCover_2k_	NN	49.515194	94.859	45.766693	93.985	45.816673	93.797
ForestCover_4k_	HSFC_NN	50.809676	0.156	50.114954	0.203	49.930028	0.203
ForestCover_4k_	NN	58.146741	186.828	56.862255	187.797	57.477009	187.859
ForestCover_8k_	HSFC_NN	57.646941	0.156	55.147941	0.203	54.793083	0.203
ForestCover_8k_	NN	64.614154	373.485	62.664934	374.5	63.834466	374.531
ForestCover_16k_	HSFC_NN	58.576569	0.188	55.662735	0.235	55.732707	0.25
ForestCover_16k_	NN	65.19892	746.328	63.269692	747.5	64.414234	747.469
ForestCover_32k_	HSFC_NN	60.545782	0.266	58.716513	0.297	58.526589	0.328
ForestCover_32k_	NN	68.472611	1487.843	64.32427	1489.047	65.353858	1489.093
ForestCover_64k_	HSFC_NN	58.601559	0.39	56.852259	0.422	58.581567	0.422
ForestCover_64k_	NN	69.167333	2971.922	58.371651	2972.906	58.811475	2973.015
ForestCover_128k_	HSFC_NN	58.696521	0.641	57.007197	0.672	58.661535	0.672
ForestCover_128k_	NN	69.537185	5952.219	56.332467	5955.594	56.467413	5953.406
ForestCover_256k_	HSFC_NN	57.267093	1.188	54.453219	1.188	57.212115	1.203
ForestCover_256k_	NN	67.443023	11916.312	49.595162	11925.297	49.910036	11879.274
ForestCover_512k_	HSFC_NN	56.897241	2.219	55.667733	2.266	55.897641	2.266
ForestCover_512k_	NN	62.240104	23616.265	54.328269	23577.36	54.523191	23575.984

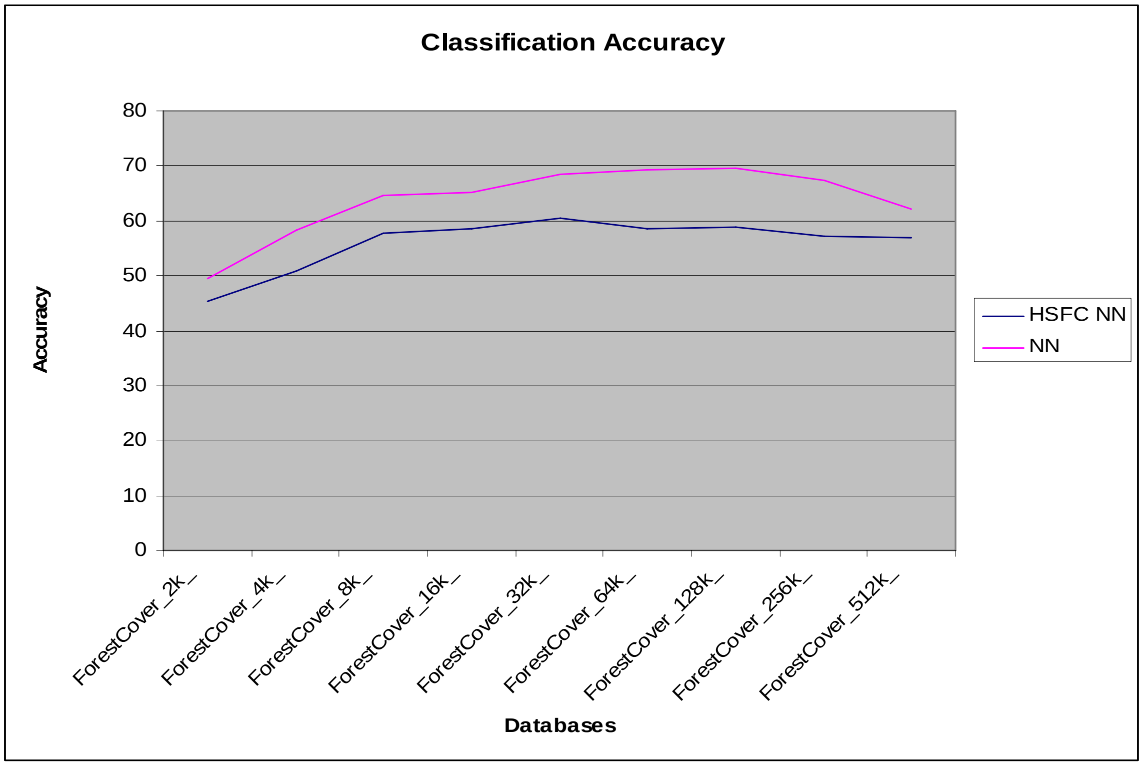

Figure 9: Forest Cover Accuracy

The results from the Forest Cover experiments show that the HSFC NN gives a dramatic speed increase, but at a cost. In most of the experiments there is a significant reduction in the accuracy of the HSFC NN in relation to the Standard NN. For, example the k-NN algorithm running on the 32k training set achieved an accuracy of 68.47% compared to the HSFC k-NN with an accuracy of 60.54%. The run time for the HSFC k-NN however is much faster than the Standard NN, at 0.266 seconds compared to 1488 seconds. The run times for each algorithm are shown in the figure below.

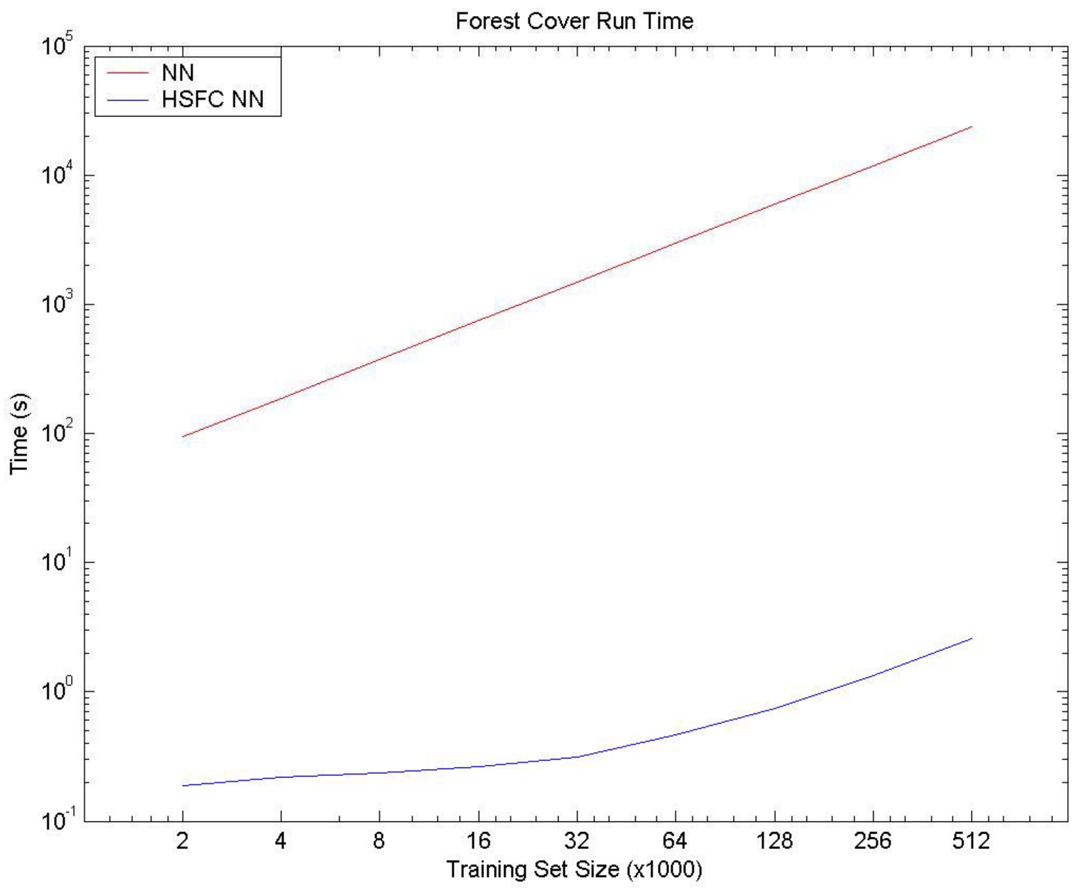

Figure 10: Forest Cover Database Run Times

In the case of the Forest Cover Database the speed increase from the HSFC NN comes at a significant reduction in the accuracy. In the next section the experiments are run again on both the small and large databases using multiple curves. Using multiple curves should decrease the number of neighbors that are missed by the HSFC NN and increase the accuracy of the HSFC NN. It is believed that this method will close the gap between the HSFC NN and the Standard NN on difficult data sets.

4.3.2 Multiple Curve HSFC NN

In this section the HSFC NN classifier is altered to use multiple curves in the classification phase. It is believed that this will increase the accuracy, and bring the performance of the HSFC NN closer to the performance of the Standard NN on difficult databases. To create multiple curves, the data is shifted by one unit towards and away from the origin. When these datasets are indexed using Butz's algorithm, it will create two extra curves that have different orders of the training points and that have saved different n-space relationships.

When using multiple curves there are a number of ways to classify the points. One method is to allow each of the three curves to classify the point independently, and then let them vote on the classification of the point. A second method, the one implemented in this section, is to find the closest neighbors from each of the curves, and use the best neighbor or neighbors from each to classify the point.

The results of the altered algorithm on the Forest Cover Database, and on selected small databases are presented in the following tables.

Table 4: Multi vs Single HSFC accuracy on the Forest Cover Database

DBName	Alg Type	One NN	Run Time	*k*-NN	Run Time	*k*-WNN	Run Time
ForestCover_2k_	Multi Curve	46.53639	0.344	43.487605	0.422	43.297681	0.438
ForestCover_2k_	Normal	47.30108	0.14	47.785886	0.234	43.932427	0.188
ForestCover_4k_	Multi Curve	52.68393	0.359	52.064174	0.406	51.064574	0.406
ForestCover_4k_	Normal	50.85966	0.156	49.445222	0.219	47.361056	0.219
ForestCover_8k_	Multi Curve	57.95682	0.422	56.597361	0.469	56.077569	0.453
ForestCover_8k_	Normal	55.69772	0.172	49.965014	0.25	49.92503	0.235
ForestCover_16k_	Multi Curve	58.30168	0.516	56.937225	0.562	56.437425	0.562
ForestCover_16k_	Normal	55.89764	0.219	56.297481	0.266	55.267893	0.266
ForestCover_32k_	Multi Curve	59.53119	0.719	58.276689	0.75	57.472011	0.735
ForestCover_32k_	Normal	57.02719	0.312	55.582767	0.328	57.027189	0.313
ForestCover_64k_	Multi Curve	58.27169	1.094	56.157537	1.125	55.437825	1.141
ForestCover_64k_	Normal	54.96801	0.453	53.253699	0.469	54.963015	0.469
ForestCover_128k_	Multi Curve	58.15174	1.89	56.902239	1.938	55.077969	1.937
ForestCover_128k_	Normal	54.95302	0.719	52.314074	0.75	54.938025	0.75
ForestCover_256k_	Multi Curve	55.20292	3.547	53.363655	3.547	51.07457	3.562
ForestCover_256k_	Normal	48.9954	1.313	47.501	1.437	48.985406	1.329
ForestCover_512k_	Multi Curve	56.72731	6.906	52.264094	6.906	49.830068	6.906
ForestCover_512k_	Normal	52.2591	2.531	50.87465	2.516	50.929628	2.579

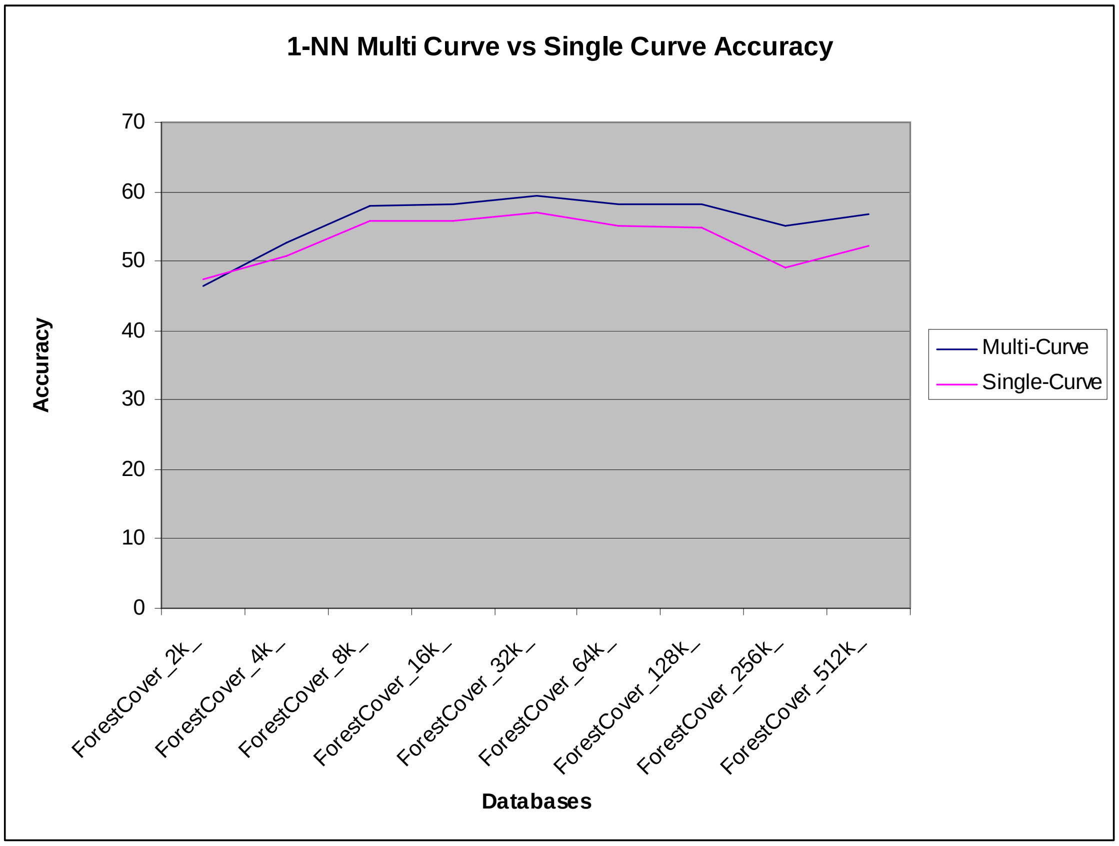

Figure 11: Multi-HSFC 1-NN vs. Single-HSFC 1-NN

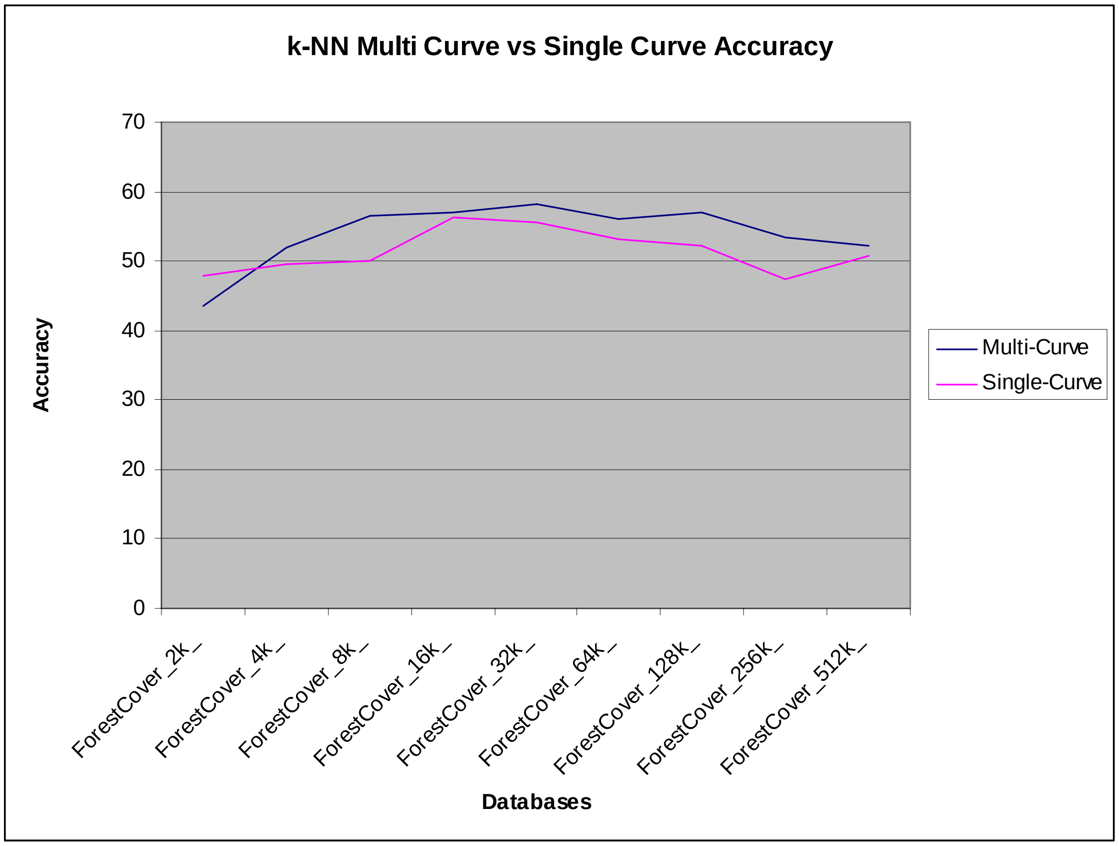

Figure 12: Multi-HSFC k-NN vs. Single-HSFC k-NN

The Figures above show that on the Forest Cover Database the Multi-HSFC approach improves the accuracy for most of the databases tested. However the increases do not bring the accuracy of the HSFC NN classifier up to the level of the Standard NN classifier for these databases. The Standard NN classifier still outperforms the HSFC NN Classifier even using the Multi-curve approach. The high dimensionality of the Forest Cover database is mostly to blame for the large deficit in accuracy. As the number of dimensions increases the number of possible neighbors within one unit from the test point increases while the number of neighbors within one unit on the HSFC remains at two. This means that as the number of dimensions increase it becomes less likely that the HSFC will produce the same nearest neighbor as the Standard NN. While using the multi curve approach did increase accuracy, in order to bring that accuracy up to the level of the Standard NN it would require using many more curves. The best chance of achieving that accuracy would be to use a curve that has been shifted towards each corner of the n-space. The problem with this is that for each curve you create you have to store another copy of the training data. On large databases like the Forest Cover that would not be practical. It is possible that the other type of multi curve NN discussed earlier could improve on the accuracy found here. The implementation of a voting curve system will be experimented some time later.

The multi-curve experiments were also run on the small databases. It was found that the accuracy of the 1-NN was only slightly affected changing very little for better and worse on different databases. However it was noted that the performance of the k-NN and Weighted k-NN suffered as a result of the multi-curve method. In general it was found that the multi-curve method used is producing the best improvements when applied to the 1-NN classifier.

Chapter 5: Conclusions on Results

The experiments conducted in the research were carried out to test the validity of using the HSFC to decrease the runtime of the Nearest Neighbor classifier. In each and every case the HSFC NN classifier outperformed the Standard Nearest Neighbor in terms of speed. As the number of training points increases the difference in run time becomes more and more pronounced. On the smaller databases, such as the Gaussian 2 class database, the HSFC classifies the database 80.06 times faster than the Standard NN, while on the larger datasets like the Forest Cover 512k database the HSFC classifies the database 10642.75 times faster reducing the classification time from 6.56 hours to 2.219 seconds. These results show that the HSFC NN successfully improved on the classification time of the Nearest Neighbor algorithm. This improvement however comes at the cost of accuracy. On low dimensionality databases this cost in accuracy is very minimal as the results on the small databases show; the HSFC maintains accuracy very close to the Standard NN in most of the small databases. However, on higher dimensionality problems, and on databases with complicated decision boundaries the HSFC accuracy falls short of the Standard NN. The results show that the accuracy of the HSFC NN can be improved by using multiple curves, but this technique does not improve the accuracy enough to match the Standard NN. The accuracy could be improved further by increasing the number of curves used to classify the data, but with each additional curve another copy of the training data must be stored. At this point it becomes a trade off between space and accuracy; higher accuracy can be achieved but at the cost of space. Several other techniques for increasing the accuracy of the HSFC NN will be discussed in the future work section.

Overall the experiments show that the HSFC is able to classify data reliably, in most cases the accuracy is comparable to the Standard NN, and in each instance it gives a significant improvement in classification time. The HSFC NN would be a good candidate for applications where speed is the most important factor. Also in low dimensional problems the HSFC NN is interchangeable with the Standard NN giving equivalent results, and improved classification time.

Chapter 6: Implementation Issues

This chapter details the implementation of the HSFC NN and the NN classifiers, and also discusses some of the issues that arose during the process. The code and scripts used in the experiments can be found in Appendix B.

The HSFC code used to convert n-dimensional data to the Hilbert index was implemented by Doug Moore of Rice University. This code will not be included in the appendix as it is interchangeable with any function that implements Butz's Algorithm. This code was used for its simple interface and straight forward implementation.

The HSFC NN and the Standard NN are both implemented in a single C++ global function. This function takes a number of parameters that are needed for each classifier to run. Most of the parameters, such as the number of neighbors *k*, the training data, and the test data are used by both classifiers. However parameters like the order of the Hilbert curve are used only by the HSFC classifier. To control which of the two classifiers is used there is a single Boolean argument passed as the first parameter. Both classifiers are Nearest Neighbor classifiers, so the code used to classify each point remains the same for both the HSFC NN and the Standard NN. The difference comes in the selection of the neighbors. Once the algorithm is started the correct neighbor selection routine is run according to the classifier that is selected. In the case of the HSFC NN the indexing and sorting of the training data happens only once, at the beginning of the algorithm, and only if the HSFC classifier is selected. After this phase is complete, the algorithm goes into the main loop. This loop runs through each one of the points in the test set. Inside this loop each point is classified, and the classification is checked against the known classification to test if the algorithm was correct. The process of classifying a test point begins with an *if* statement that checks to see which of the two classifiers is to be used. In the case of the HSFC the test point is converted to a Hilbert index, and then a modified binary search is run on the training set to find the test index. The binary search was modified to return the index of either an exact match or the index of the point just after where the test index would fall. This search would return an index on the training array. Starting with this index and moving out away from the test point, neighbors are selected and stored in the neighbor array. In the case of the Standard NN the distance from the test point to all the other points in the training set are calculated and the information, pertaining to the training points of the smallest *k* distances, is stored in the neighbor array. After this point the algorithms are the same. The neighbor array is used to classify the test point according to the type of classification being done; 1-NN, k-NN, or Weighted k-NN.

The implementation of the Standard NN is straight forward and the neighbor selection process is simply finding the smallest distances among the training points. As such, once this section was coded there was no need for modifications. The HSFC NN neighbor selection however is not as straight forward, and as such a few modifications were made to improve the performance of the classifier. When the test point's location on the Hilbert curve has been found, there are two possible locations for its nearest neighbor either before or after it on the curve. In the original implementation the first neighbor was selected as the index that was returned by the search, the next neighbor was selected as the index that came before the test point, the rest of the neighbors were selected alternating before and after the test index until the number of neighbors was reached. This method did not take into account the distance of these neighbors from the test point on the curve. To clarify, if the number of neighbors *k* was 6, then 3 neighbors were selected before the test point on the curve and 3 neighbors were selected from after the test point on the curve. This method is not the best choice because we are not guaranteed that the HSFC is full. There could be any number of missing indexes between the test point and the training points just before and after the test point in the training array. The neighbor selection method was altered to take into account the distances on the curve. In the new method the 1st neighbor is selected as the closer of the test point's first neighbors on either side. According to which one is chosen a counter is incremented and the loosing point is tested against the next point further out from the test point on the winning side. This method increased the accuracy of the classifier slightly by ensuring that only the closest points on the Hilbert curve were selected as neighbors.

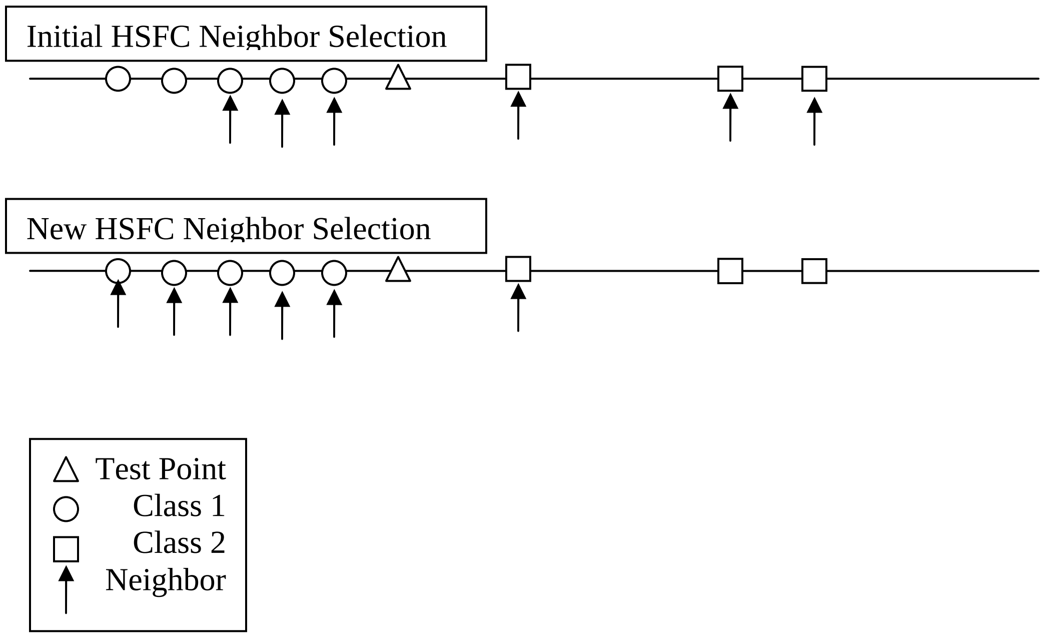

Figure 13: HSFC Neighbor Selection

The experiments were run in MATLAB; this was accomplished by compiling the algorithm into a MATLAB DLL. This was the most difficult process of the implementation. In order to create the MATLAB DLL requires an interface. This interface file is used to talk between MATLAB and the NN_Alg function. It defines the function that is called from MATLAB, and also does all of the conversions between the MATLAB Array types to C arrays, and it takes care of error checking. When the function is called from MATLAB, the parameters are passed to the interface file, where they are converted to the correct types and checked to make sure that they are within the accepted ranges. Then the interface calls the NN_Alg function with the converted parameters. After the algorithm has finished the interface file returns the accuracy and the results back to MATLAB. The implementation of the MATLAB DLL, while adding some complexity to the process made it possible to quickly run experiments and generate data using MATLAB scripts. It also made handling the databases easier by removing the need for C++ file access routines.

The largest issue that arose from the MATLAB implementation was the fact that once the algorithm had run MATLAB did not free up the memory used inside the function. This meant that after many experiments had been run the computer would start to slow down because MATLAB was taking up more and more memory. This was solved by changing the C++ algorithm to take care of freeing up the memory manually.

In general the implementation of these algorithms is fairly simple. The actual code is straight forward and simple to understand. This makes these classifiers much easier to implement and run than other more complicated classifiers such as neural networks.

Chapter 7: Future Research

During the course of this research many ideas for future work were brought to light. There are a few techniques to improve the accuracy of the HSFC NN classifier that have yet to be implemented, and an idea for a new type of classifier based on the HSFC similar to the patterns of the ARTMAP Neural Network is being formulated (Carpenter,1992). Also there are other classification techniques that could benefit from reducing the dimensionality of the problem, such as the Probabilistic Neural Network (Specht, 1990).

In this research one technique using multiple HSFC's was implemented and tested. It was shown that this technique improved the accuracy for the Forest Cover database, but had mixed results with the smaller databases. Other multi-curve methods are also available and could be implemented in the future. One such method discussed earlier is to allow each individual curve to classify the test point independently, and then allow them to vote on the final classification of the test point. This method is desirable because this means that the original results obtained would still be intact within the original curve. This technique is therefore less likely to perform worse than the single curve implementation. It is hoped that this technique would improve the accuracy in the k-NN and Weighted k-NN as well. Another multi-curve technique designed to improve accuracy on high dimensionality databases, would be to split the dimensions of the database into groups and use a different HSFC for each group of attributes. For example, on the Forest Cover Database, the 12 attributes would be split into groups of 4, and each of these three groups would be indexed onto separate curves. The curves would then vote on the classification of each test point. This method would reduce the amount of spatial information that is lost from the original data.

The Hilbert Decision Space Classifier is based on the clustering properties of the HSFC. The idea is to create ranges on the HSFC that are mapped to a certain class. These ranges would be similar to the patterns in ARTMAP Neural Networks. The training data would be used to create these ranges, and then once the training is complete the training data can be discarded. This classifier will be very fast because most of the work will be done during the training phase. Once the training is complete the test points can be classified quickly by finding the correct range. The ranges will all be in one dimension so the correct range can be found using a binary search which again is very computationally efficient. Also the number of ranges should be less than the number of training points so the binary search will be over a smaller array. The major benefit of this technique, however, would be the ability to discard the training data, and the memory savings that would bring, especially on large databases like the Forest Cover. This new classifier would be much more complicated to implement than the HSFC NN. The biggest challenge in designing this technique would be to define the training phase. There are many issues that must be taken into consideration when creating the ranges, such as, what to do about points that fall into a range but do not agree on the classification. Should a new range be created in the middle of the current range, or should the point be thrown out as noise. In the end it is likely that the training phase will require multiple passes, with the first pass taking in all of the training data, and the subsequent passes weeding out un-needed ranges. The Hilbert Decision Space Classifier (HDSC) will likely be the focus of my future research; if this technique maintains the accuracy of the HSFC NN along with the added benefit of discarding the training data, it could turn out to be a very robust classifier. The following figure shows a possible selection of ranges on the IRIS database. The ranges colored blue would be class 1, and the ranges colored red would be class 2. This range selection would be using very loose parameters as can be seen by the number of points that are in the wrong ranges.

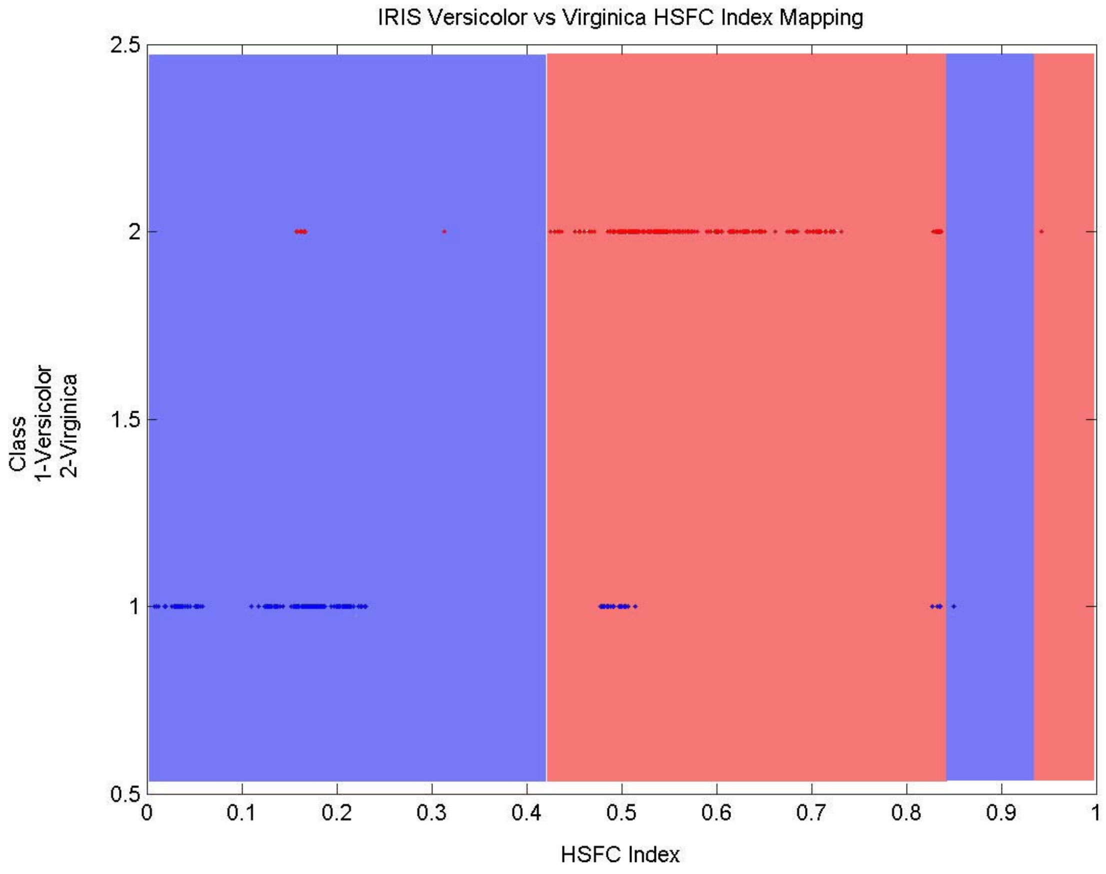

Figure 14: Hilbert Decision Space Classifier Example

Other classifiers that would benefit from a reduction in dimensionality could also be implemented using the HSFC. One such classifier is the Probabilistic Neural Network. Implementing the PNN using the HSFC would be much more complicated than the

Nearest Neighbor, and as such would require more time and energy. The PNN works by estimating the probability distribution of the training data, and makes is classification based on the Bayesian classifier. Further research is required to determine how to implement the PNN using the HSFC.

One last avenue that might be pursued would be to develop a technique for estimating the expected accuracy of a given database. When testing on generated databases the optimal accuracy is known, however, on natural databases, unless all the statistical information about the data is known, it is difficult to know what the optimal accuracy would be. When multi-dimensional data is mapped onto the Hilbert curve it is possible to see relationships between the classes that are impossible to graph in multiple dimensions. We hope to develop a technique to estimate the expected accuracy of data by mapping the data to the Hilbert curve and determining the overlap of the classes. This would be useful to many researchers working on Natural databases.

The Hilbert Space Filling Curve has many qualities that make is useful in classification algorithms. It reduces the dimensionality of the data while still maintaining some of the spatial relationships, as well as naturally clustering the data. These properties will undoubtedly lead to many new applications for the HSFC.

References

Arthur R. Butz., “Alternative Algorithm for Hilbert's Space-Filling Curve”. *IEEE Transactions on Computers*, 20:424-426, April 1971.

Carpenter, G. A., Grossberg, S., Markuzon, N., Reynolds, J. H., "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multi-dimensional maps," IEEE Transactions on Neural Networks, Vol. 3, No. 5, pp. 698-713, 1992.

Castro, Jose; Georgiopoulos, Michael; Demara, Ronald; Gonzalez, Avelino; “Data-Partitioning using the Hilbert Space Filling Curves: Effect on the Speed of Convergence of Fuzzy ARTMAP for Large Database Problems,” *Neural Networks*; accepted for publication.,2004

Friedman, J.; Baskett, F.; Shustek, L. “An Algorithm for finding nearest neighbors,” *IEEE Transactions on Computers*, Vol. 24, 1000-1006, 1975

T. Cover, P. Hart, “Nearest Neighbor Pattern Classification,” *IEEE Trans. on Information Theory*, IT – 13:21-27, 1967

J.K.Lawder; P.J.H.King. “Using Space-filling Curves for Multi-dimensional Indexing.” *Advances in Databases: proceedings of the 17th British National Conference on Databases (BNCOD 17)*, volume 1832 of *Lecture Notes in Computer Science*, pages 20 - 35. Springer Verlag, July 2000

J.K.Lawder; P.J.H.King. Using State Diagrams for Hilbert Curve Mappings. *International Journal of Computer Mathematics 78(3): 327-342 (2001)* (formerly Research Report BBKCS-00-02 or JL2/00, August 2000)

J.K.Lawder. "Calculation of Mappings Between One and n-dimensional Values Using the Hilbert Space-filling Curve." Research Report BBKCS-00-01 (formerly JL1/00), August 2000 http://www.dcs.bbk.ac.uk/~jkl/publications.html

Skubalska-Rafajlowicz, E.; Krzyzak, A., "Fast k-NN classification rule using metric on space-filling curves," *Pattern Recognition, 1996, Proceedings of the 13th International Conference on*, Volume: 2, Pages: 121 – 125, 25-29 Aug. 1996

Specht, D. F., "Probabilistic Neural Networks", Neural Networks, Vol. 3, pp. 109-118, 1990.