8-2024

Ensemble Machine Learning at the Edge Using the Codec Classifier Structure and Weak Learners Guided by Mutual Information

AJ Beckwith
Utah State University

ENSEMBLE MACHINE LEARNING AT THE EDGE USING THE CODEC CLASSIFIER STRUCTURE AND WEAK LEARNERS GUIDED BY MUTUAL INFORMATION

by

AJ Beckwith

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Jacob Gunther, Ph.D. Major Professor	Todd K. Moon, Ph.D. Committee Member
Kevin Moon, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2024

ABSTRACT

Ensemble Machine Learning At The Edge Using The Codec
Classifier Structure And Weak Learners Guided By
Mutual Information

by

AJ Beckwith, Master of Science
Utah State University, 2024

Major Professor: Jacob Gunther, Ph.D.
Department: Electrical and Computer Engineering

The Codec Classifier is a low-computation, low-memory tree ensemble method that dramatically improves feasibility of image classification on resource-constrained edge devices. It achieves advantages over other tree ensemble methods due the separation of encoder and decoder tasks in the classifier. The encoder partitions feature space, and the decoder labels the regions in the partition. This functional separation of tasks enables the encoder design (partitioning) to be guided by maximizing the mutual information (MI) between class labels and the features (i.e. the encoded representation of the data) without regard to the error performance of the classifier. Experiments show maximizing MI leads to sequential partitioning of feature space that is more efficient than additive classifier models such as AdaBoost. The design results in gray-coded partitions in which adjacent regions are addressed by codewords that differ in only one bit. This novelty affords classification insights not available with other methods. The method is applied to binary classification (face detection) and multiclass classification (MNIST digits) problems.

(44 pages)

PUBLIC ABSTRACT

Ensemble Machine Learning At The Edge Using The Codec
Classifier Structure And Weak Learners Guided By
Mutual Information

AJ Beckwith

The Codec Classifier is a low-computation, low-memory tree ensemble method that dramatically improves feasibility of image classification on resource-constrained edge devices. It achieves advantages over other tree ensemble methods due the separation of encoder and decoder tasks in the classifier. The encoder partitions feature space, and the decoder labels the regions in the partition. This functional separation of tasks enables the encoder design (partitioning) to be guided by maximizing the mutual information (MI) between class labels and the features (i.e. the encoded representation of the data) without regard to the error performance of the classifier. Experiments show maximizing MI leads to sequential partitioning of feature space that is more efficient than additive classifier models such as AdaBoost. The design results in gray-coded partitions in which adjacent regions are addressed by codewords that differ in only one bit. This novelty affords classification insights not available with other methods. The method is applied to binary classification (face detection) and multiclass classification (MNIST digits) problems.

To my father.

CONTENTS

LIST OF FIGURES

ACRONYMS

MI	Mutual Information
MNIST	Modified National Institute of Standards and Technology database
FFT	Fast Fourier Transform
KDE	Kernel Density Estimation
MSE	Mean Squared Error
KSG	Kraskov–Stögbauer–Grassberger
RGB	Red Green Blue
PASS	Pictures without humAns for Self-Supervised Pretraining
FPGA	Field Programmable Gate Arrays
GPU	Graphics Processing Unit
CPU	Central Processing Unit

CHAPTER 1

INTRODUCTION

The efficiency of accurate, flexible machine learning methods are increasingly important as applications of machine learning are rising. Edge devices like satellites, drones, and hand-helds can benefit from local machine learning. Machine learning at the edge enables classification without the need for large bandwidths or memory. The data collected from sensors and cameras can, in real-time, be used for training and inference. Requiring only the storage of low-memory outputs and their transmission on low-capacity communication channels.

Neural networks [1] are by far the most popular structure for classification because of their accuracy and adaptability. But what they gain in classification ability they lose in computation time and model complexity.

1.1 Classification at the Edge

The resources necessary to deploy a software-based classifier include electric power, computer memory, computational hardware, and data communication. When considering applications on the edge, such as classification on a satellite, a drone, and hand-held devices, all of these resources are severely limited. Neural networks generally require millions or billions of arithmetic operations per inference, which consumes considerable amounts of all resources previously enumerated. Every node in every layer of a deep fully-connected neural network must store a n-element vector where n is equal to the number of inputs for that layer. Convolutional neural networks gain efficiency through the use of kernels, but they also use many kernels per layer and require convolution operations which are expensive, even when exploiting frequency domain processing using the FFT. The layered structure of a neural network also limits the number of parallel computations that can be performed. In contrast to the heavy computational and memory burden of neural networks,

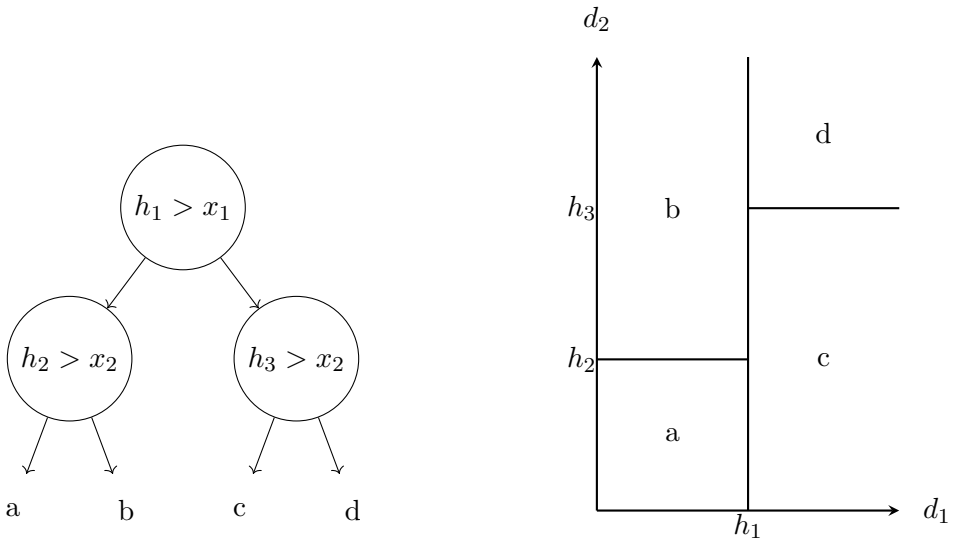
tree ensembles are simple classification methods that require little computation and memory when compared to similarly capable neural networks [2]. Tree ensembles may be limited in their classification ability. The ideal tree ensemble has a minimum number of trees to describe the data distribution in feature space so that all regions in the feature space contain training data from only one class. This ideal places an upper limit on the number of computations necessary for classification since the computation time is proportional to the number of trees. Common methods for aggregating ensemble outputs is majority vote or weighted sum. Both of these methods suffer from classification error even when a region is pure due to aggregation of the tree outputs. The codec classifier aims to alleviate the drawbacks of traditional tree ensemble methods. It finds a minimal tree description of a feature space and can independently reference the regions in the description.

1.2 Tree Ensemble

Ensemble methods aggregate the outputs of multiple weak learners into a comprehensive prediction. Among ensemble methods, the type of weak learner, the method for training, and the method for aggregation vary significantly [3]. The type of weak learner used in this thesis is a tree. Fig. 1.1 shows the general structure of a tree with depth 2. Decisions in the tree require the computation of an inequality where h_i is the i^{th} threshold and x_i is the input in the dimension d_i. The outputs of the tree correspond to regions in the feature space. Some tree ensembles exclusively use trees of depth 1, called stumps. Fig. 1.2 shows how space is partitioned using three stumps. The thresholds of stumps don't terminate at other thresholds as with the depth 2 tree. The trees used in this thesis are stumps.

1.3 Adaboost

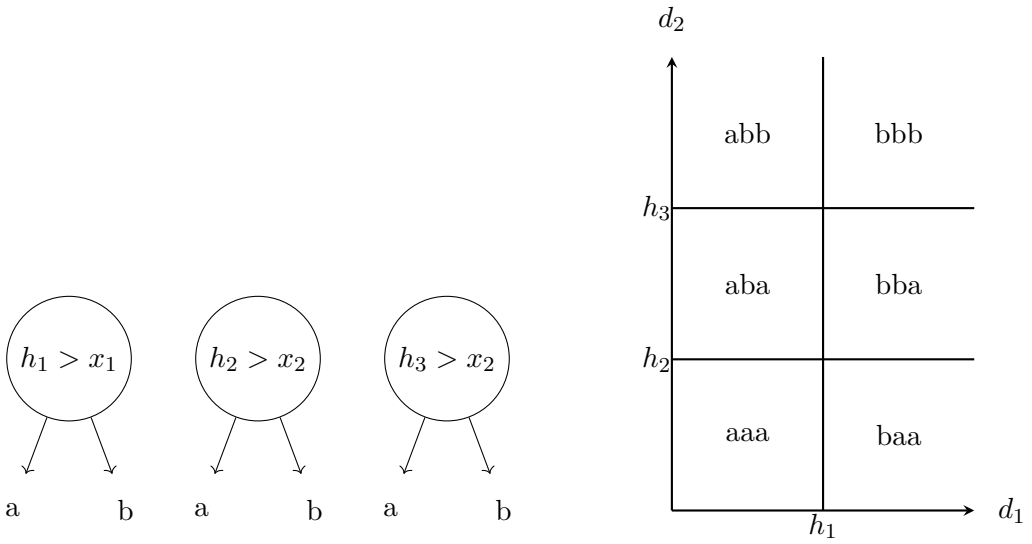
Two common training methods for tree ensembles are boosting [4] and bagging [5]. Adaboost [6] is a boosting method that sequentially trains trees. As trees are trained the weights of points to be classified are adjusted to emphasize misclassified points. This



(a) Tree with Depth 2

(b) Space Partition of Tree in (a)

Fig. 1.1: Depth 2 Tree and its Space Partition



(a) 3 Stumps

(b) Space Partition of Trees in (a)

Fig. 1.2: 3 Stumps and their Space Partition

method is an additive model [7] and can be described by

$$f(x) = \sum_{i=0}^{N} w_i h_i(x), \tag{1.1}$$

where N is the number of weak learners, w_i is the weight, and $h_i(x)$ is a weak learner.

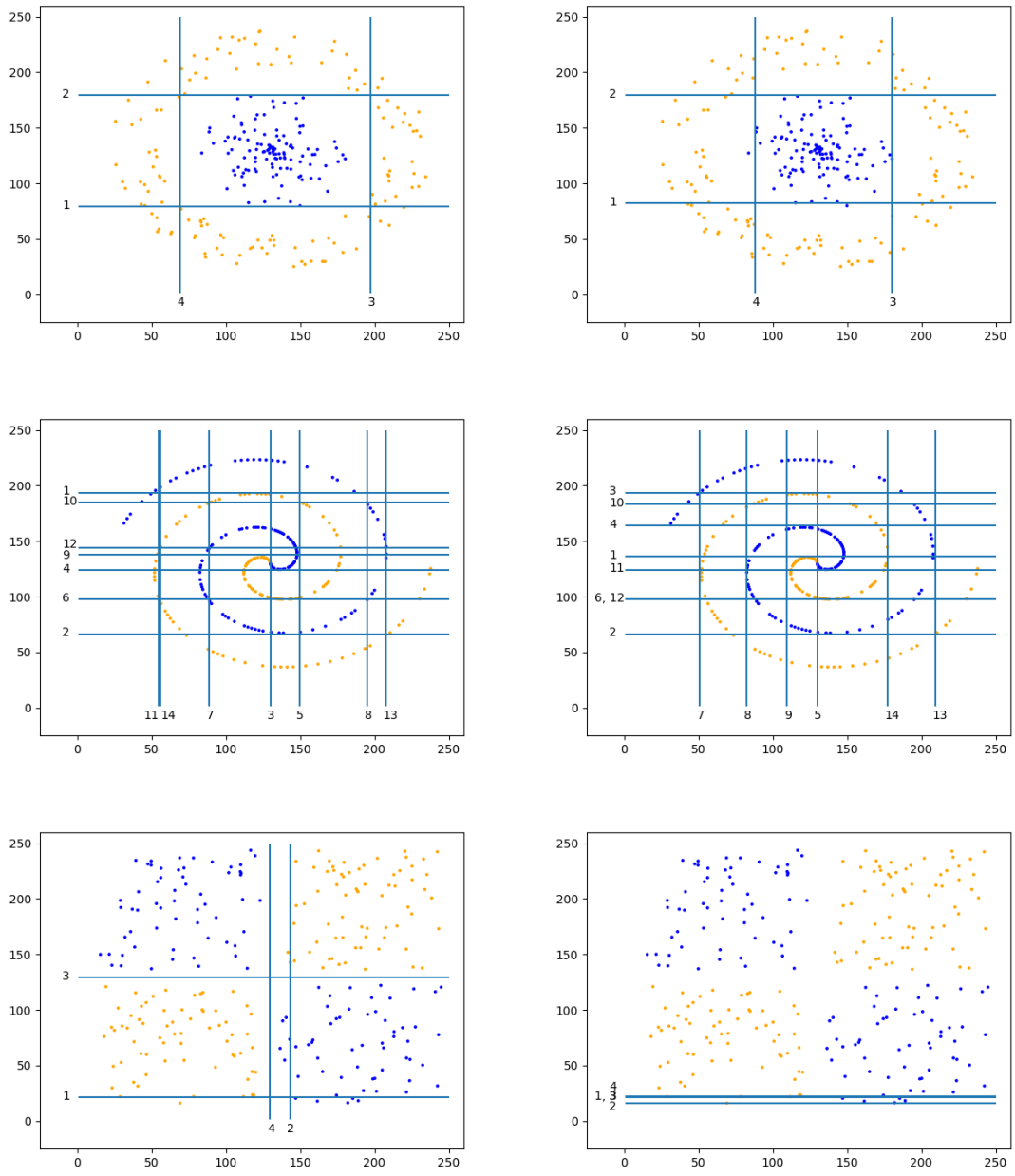
Fig. 1.3 shows three simple classification problems and the tree threshold positions chosen by Adaboost and the method developed in this thesis, the codec classifier. The order that the trees were learned in is also indicated next to each partition. The codec classifier trained trees until 0% classification error on the training data. The Adaboost classifier was allowed to train an equal number of trees. The classification accuracy for the Adaboost eye, spiral, and cube data are 99.2%, 97.6%, and 55.2%, respectively. The Adaboost classifier performed only slightly worse than the codec classifier on the eye and spiral data. On the cube data it performed significantly worse. Even with several more trees the Adaboost classifier still performs badly on the cube dataset. It is also important to note that the Adaboost classifier duplicates tree positions on the spiral and cube datasets. Duplicated trees cannot improve the partition of the feature space but still increase the complexity and computation time of the classifier.

1.4 Mutual Information

Mutual information is defined as the amount of information one random variable contains about another random variable [8] and is defined by,

$$I(X;Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}. \tag{1.2}$$

As this definition shows, the probability density function of the random variables must be known for calculation of mutual information. Several methods exist for estimating mutual information on non-parametric data [9]. Moon et al. [10] propose a Kernel Density Estimation method for probability density estimation. KDE generally features superiority over traditional histogram-bin methods. Some included advantages are better MSE convergence



(a) Codec Classifier

(b) AdaBoost Classifier

Fig. 1.3: Tree Threshold and Dimension Comparison

rate, origin insensitivity, and options for varied window shapes. Other methods include KSG [11], Parzen Window Density Estimation [12], and K-Nearest Neighbor [13]. While some of these methods can approach parametric mean square error rate of convergence given the appropriate conditions, most of them require significant computational overhead.

Noshad et al. [9] proposes a reduced complexity mutual information estimator. This estimator, called the ensemble dependency graph estimator, features desirable structure that works well for approximating mutual information between the data labels and the codeword representation of the data for the codec classifier. This is the method used for training the encoder in the codec classifier presented in the next chapter.

CHAPTER 2

The Codec Classifier

2.1 Architecture

The codec classifier is inspired by a simple digital communication system. It consists of two processing stages: (1) an encoder, and (2) a decoder. The encoder inputs an image and assigns a binary codeword. This codeword is then decoded to a class label by the decoder.

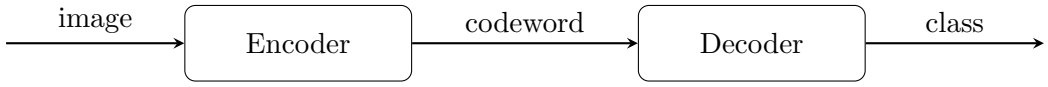


Fig. 2.1: Codec Classifier Structure

2.1.1 Encoder

The encoded representation of an image, i.e. a codeword, is determined by concatenating the outputs of the tree ensemble, where every tree outputs either a 1 or 0. All trees are decision stumps, and each tree output populates one position in the codeword bit string so that all codewords have equal length. Fig. 2.2 shows the resultant codewords when a 2-dimensional space is partitioned by the trees t_1, t_2, and t_3. A tree t_i has an output o_i, threshold h_i, and operates in a dimension d_i of the input, i.e. d_i is a pixel index. The codewords are formed as $o_1 o_2 o_3$. o_i is 1 if the input value in dimension d_i is greater than h_i and 0 otherwise. An ensemble of trees partitions the input feature space into rectangular regions, and it is worth noting that the regions of feature space are gray-coded so that there is a relationship between the Hamming distance of two codewords and the Euclidean distance between the centroids of the rectangles that those codewords represent. There exist codewords that have no associated rectangle in the feature space. However, the encoder

can only produce codewords that address regions in the partition.

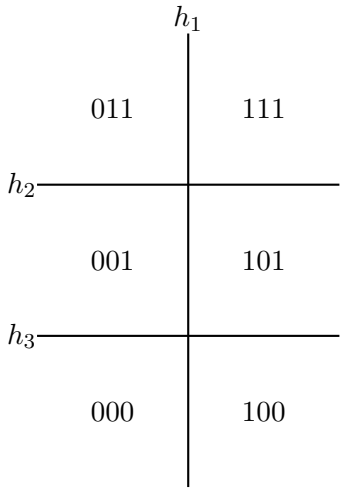
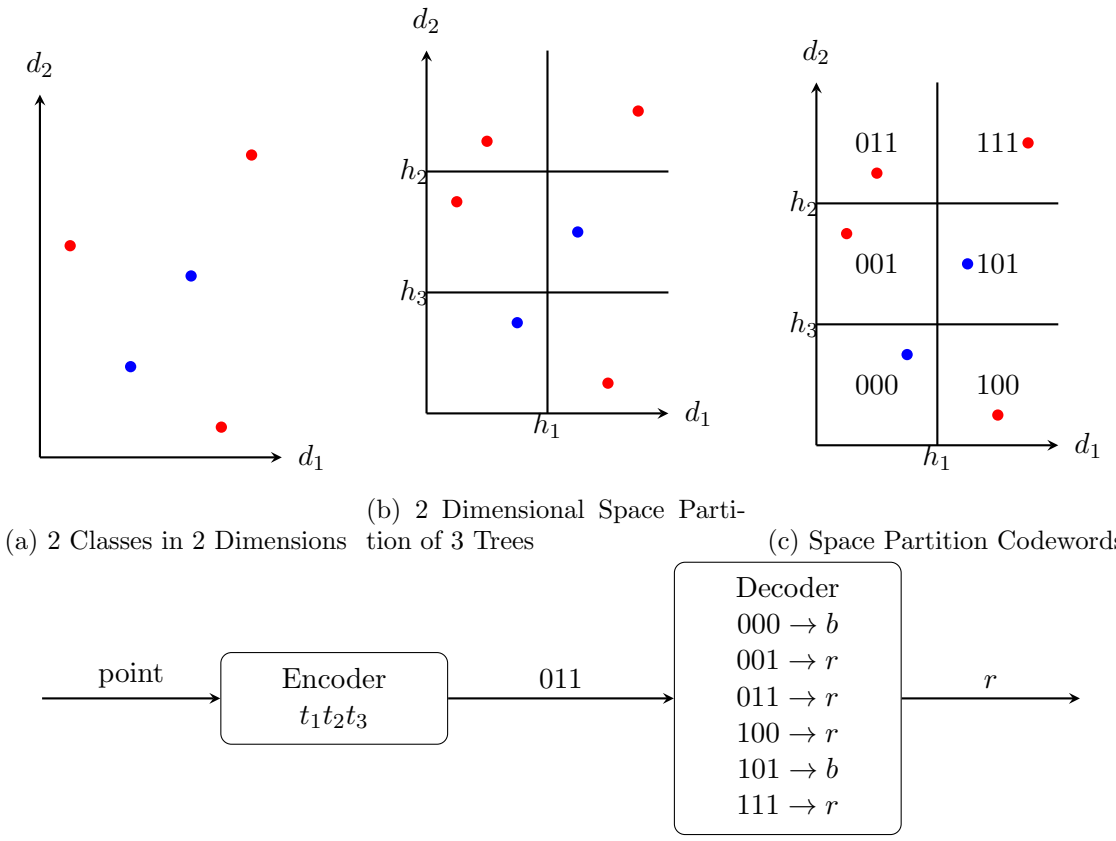


Fig. 2.2: 2 Dimensional Space partition of three trees

Encoding can be accomplished very efficiently allowing real-time implementation on low-cost or low-power hardware. The computation required per tree is evaluating a single inequality making the total computation required for an ensemble with N trees N inequalities. For serial data streams the order in which trees are evaluated can be made to match the order in which data is received.

2.1.2 Decoder

The decoder converts a codeword into a class label. Every codeword represents or addresses a unique rectangle in the feature space. The label associated with a region is learned from training data. The training set is encoded to codewords or rectangles. The label for a rectangle is the most frequently occurring label of training data falling in that rectangle. Alternatively, if a soft output is desired, then relative frequency counts can be saved in each rectangle. The association between codewords and labels is stored by the decoder as (codeword, label) pairs using a data structure such as an array, dictionary, or unordered map. To classify an image, it is first encoded to a codeword. Then the codeword is matched to a (codeword, label) pair. Fig. 2.3 demonstrates the process of classifying a



(a) 2 Classes in 2 Dimensions

(b) 2 Dimensional Space Partition of 3 Trees

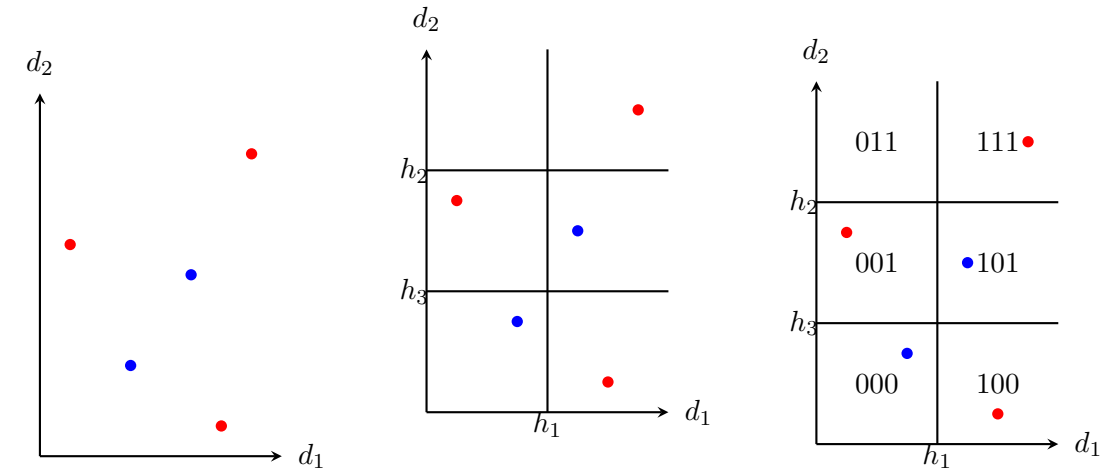
(c) Space Partition Codewords

(d) Codec Classifier Structure with Codeword-Label Pairs in Decoder

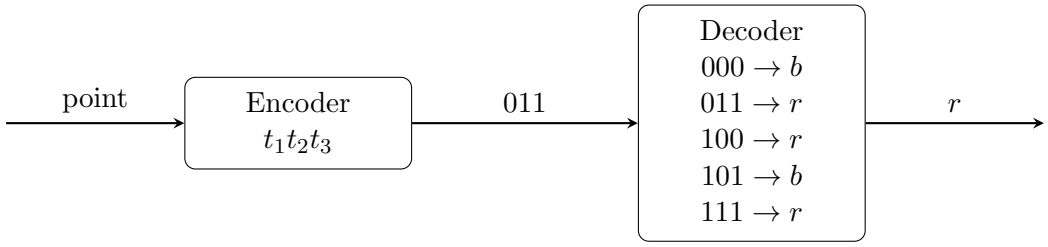
Fig. 2.3: Decoding a Point in a Labeled Region

point that falls above both h_2 and h_3 and to the left of h_1. The spaces are assigned the classes r and b as described by the decoder.

The training data are not guaranteed to populate all the rectangles constructed during the process of training trees for the encoder. Therefore, it is likely that during inference images may be mapped to a codeword corresponding to a rectangle that has not been labeled by the training dataset. In this case, other decoding rules must be used. One such decoding rule is to take the label of the stored codeword with the smallest Hamming distance from the test codeword, or the most frequent label occurring among stored codewords with the minimal Hamming distance from the test codeword. This process is described in Fig. 2.4. The decoder only contains (codeword, label) pairs from the training set.



(a) 2 Classes in 2 Dimensions (b) 2 Dimensional Space Partition of 3 Trees (c) Space Partition Codewords



(d) Codec Classifier Structure with Codeword-Label Pairs in Decoder

Fig. 2.4: Decoding a Point in an Unlabeled Region using Hamming Distance

2.2 Mutual Information

Each tree (or stump) t_i is made up by a dimension it operates in d_i and a threshold h_i where it splits the feature space in that dimension. Selecting the dimension and threshold is crucial for an ensemble that yields good predictions on the training data. Ensemble learning methods train trees in parallel, where order does not matter, or they train trees sequentially, where order does matter. Parallel examples include all forms of bagging. Boosting is sequential because the selection of a tree's threshold and dimension affect the selection of all subsequent trees' thresholds and dimensions.

To design the encoder, a sequential method that seeks to maximize the mutual information between the set of codewords and the class labels in the partition is proposed. The process of adding trees to the ensemble is continued until all regions in the partition become pure. A pure partition contains no more than one class. A set of equations that approximates the calculation of mutual information using data [9] are,

$$w_i = \frac{N_i}{N}, \tag{2.1}$$

$$w_j = \frac{M_j}{N}, \tag{2.2}$$

$$w_{ij} = \frac{N_{ij}N}{N_i M_j}, \tag{2.3}$$

$$g(x) = x\log(x), \tag{2.4}$$

$$MI = \sum_{i=1}^{I}\sum_{j=1}^{J} w_i w_j g(w_{ij}), \tag{2.5}$$

where I and J are the number of rectangles populated by data and the number of classes, respectively, N is the total number of training data used in the calculation, M_j is a count of the images with the j^{th} class, N_i is a count of the images falling in the i^{th} rectangle, and N_{ij} is the number of images with the j^{th} class in the i^{th} rectangle. This method may be applied in both binary and multi-class classification problems. Fig. 2.5 illustrates the relationship between N_i, M_j, and N_{ij} as a graph.

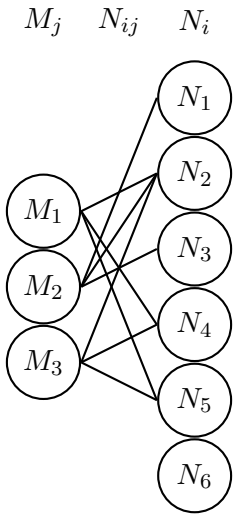


Fig. 2.5: Dependency Graph with 3 Classes and 6 Codewords

2.3 Efficient Training Implementation

For each tree, every (dimension, threshold) combination has to be tried to find the best mutual information score. This calculation has significant computation requirements especially for large data sets, data sets with images containing many pixels, or data sets whose images have pixel values that vary significantly. To compute MI efficiently a C++ program has been developed.

The MI equation is first simplified to

$$MI = \sum_{i=1}^{I} \sum_{j=1}^{J} \frac{N_{ij}}{N} \log \frac{N N_{ij}}{N_i M_j}, \tag{2.6}$$

to remove unnecessary operations.

The MI calculation requires that for each image, its class and current codeword be stored. To facilitate efficient use of memory and to optimize the computation algorithm each image is stored in the C++ struct *pointNode*. *pointNode* contains the image, its label, and a pointer to type *pointNode*.

A *codewords* array of type *pointNode*$*$ and with size equal to the number of images is created. At the start of the program all images are contained in a linked list with the head

in the first position of *codewords*. Fig. 2.6 illustrates the initial structure.

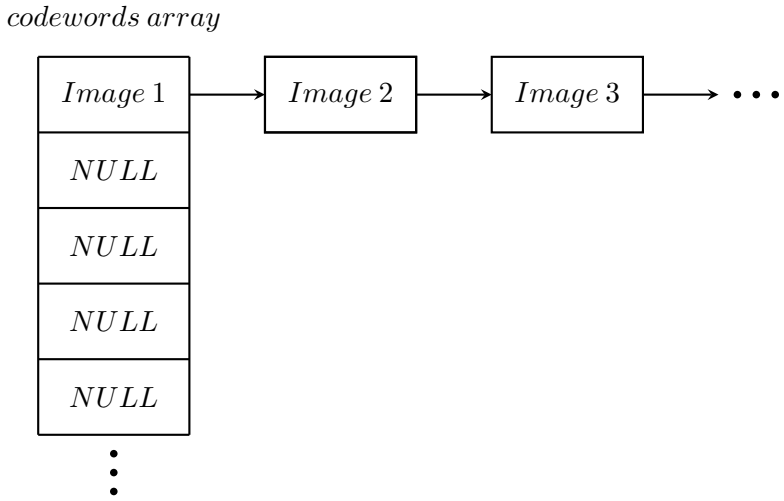


Fig. 2.6: Image Storage Structure to Compute Mutual Information

Each position in the *codewords* array represents a different codeword. Initially, before any trees are trained, every image has the same codeword. Hence, they are all linked to the first position. The number of populated positions in *codewords* is equal to the number of codewords.

At each populated position of *codewords* the algorithm moves down the linked list keeping track of how the images with that codeword would be segregated given the proposed threshold and pixel. The equation used to find the MI for the linked images of a specific codeword is given as

$$MI = \sum_{j=1}^{J} \frac{N_{1j}}{N} \log \frac{N N_{1j}}{N_1 M_j} + \frac{N_{2j}}{N} \log \frac{N N_{2j}}{N_2 M_j}, \tag{2.7}$$

where N and M_j are constant, N_1 and N_2 are a count of the images that fall below and above the threshold respectively, and N_{1j} and N_{2j} are counts of the images that fall below and above the threshold with the label j.

This process is repeated for every codeword and the MI values are accumulated. Once the total MI is found it is compared to the MI found for other thresholds and pixels. The

threshold and pixel with the greatest MI is accepted as the best placement and a new tree is trained.

When a tree is trained each linked list in the *codewords* array is segregated based on this new tree. For each group of linked images, the images that fall below the threshold of the new trees threshold keep their position in the *codewords* array. Images that fall above the threshold are moved to the next empty position in the array. In this way images with the same codeword are grouped together.

Previous implementations required significantly more computation and memory. In a two-step procedure data structures were first populated with the correct edge and node information for a given threshold and dimension. This required two nested loops with both loops having an upper limit equal to the number of data. Then using the information computed in the first step, the second step would compute the mutual information. This step also has two nested loops. One with an upper limit equal to the number of codewords and the other with an upper limit equal to the number of classes. For N equal to the number of data, W equal to the number of codewords, and C equal to the number of classes the number of loops necessary for each threshold, dimension pair is $\frac{NW}{2} + WC$. The upper limit on the number of codewords is $W \leq min(N, 2^i)$ where i is the number of trees. The efficient implementations loop complexity, $N + WC$, is far more robust to the exponentially increasing number of codewords. Additionally, the efficient implementation requires significantly less computation and variable copies in each loop. Fig. 2.7 shows the reduction in loop complexity as a function of tree number for the three datasets used in this thesis.

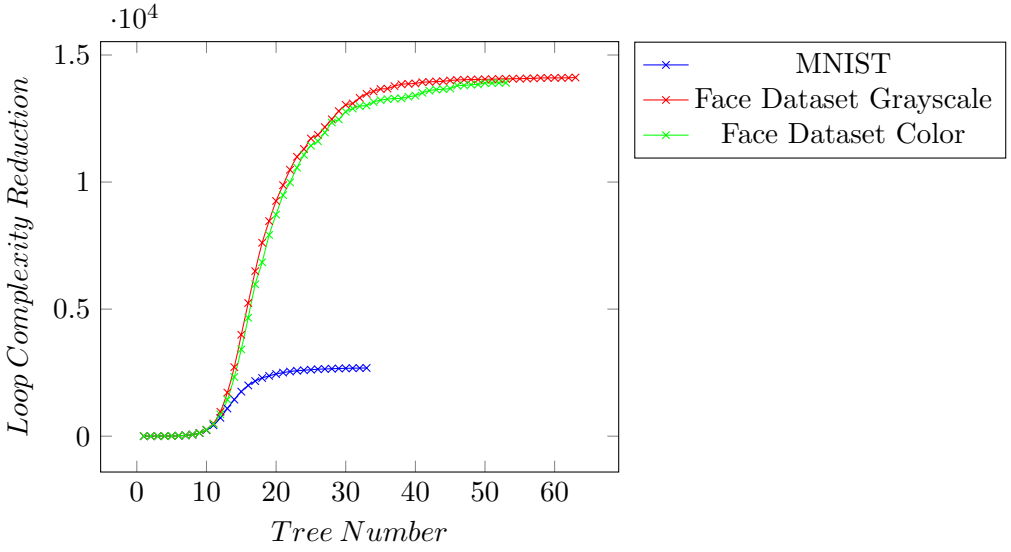


Fig. 2.7: Loop Complexity Comparison

CHAPTER 3

Results

3.1 MNIST

The MNIST dataset contains images of handwritten grayscale digits and is commonly used for machine learning training and testing. It has 60,000 images in the training set and 10,000 images in the test set. Each image has 784 pixels. Using mutual information trees were trained on the MNIST dataset until all regions were pure. With 34 trees the classification error of the training set was driven to 0. Fig. 3.1 shows the classification error as a function of the number of trees used in the encoder part of the classifier.

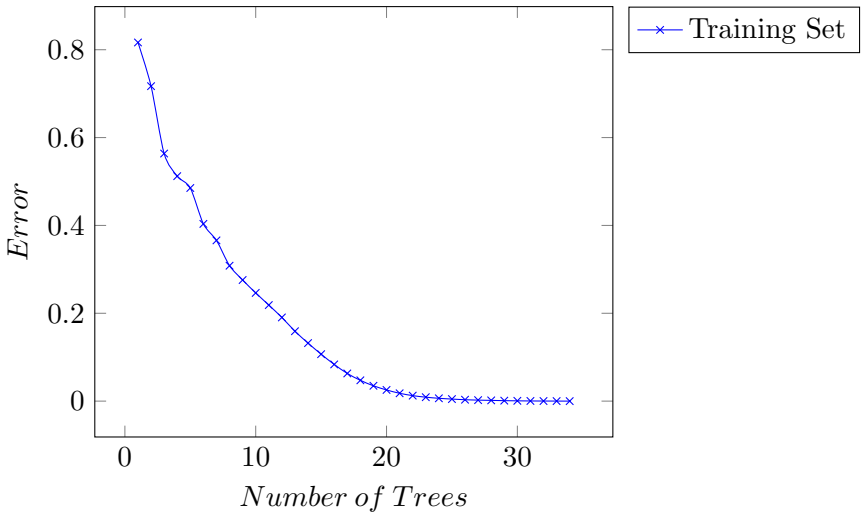


Fig. 3.1: Training Set Performance for MNIST

Fig. 3.2 includes the test set error as well as the miss and fault rates. A miss occurs when there is no matching codeword in the training set. A fault means a matching codeword was found but that the class was incorrect.

While the model was well suited for classifying the training set it did not perform

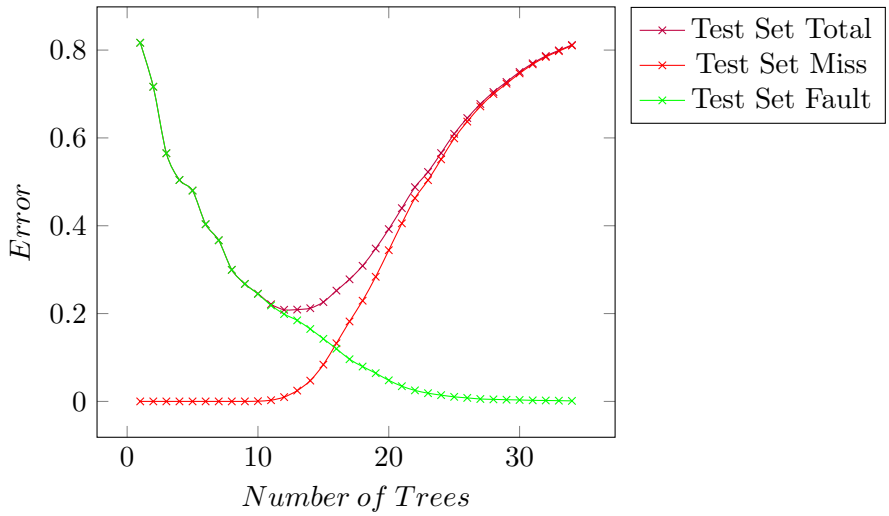


Fig. 3.2: Test Set Performance and Contributing Factors for MNIST

well on the test set. For high tree counts the majority of error occurs because there is no matching codeword. The images in the test set fell into rectangles that the images in the training set did not fall into.

This problem has been mitigated using Hamming distance. If an image doesn't fall into a labeled partition, it is given the label of the closest partition according to Hamming distance. Fig. 3.3 shows the results of using Hamming distance to label unlabeled partitions. It is remarkable that a classifier with 34 decision stumps achieves about 10% error rate on a ten class classification problem.

3.2 Face Detection

Images of faces were collected from the facescrub dataset. These images are 50×50 pixels and RGB color. 530 individuals are represented with a total of 45762 face images. 41427 images are in the training set and 4335 images are in the test set. Non-face images were sourced from the PASS dataset which contains $1,439,589$ non-human images. The first 48329 images from the PASS dataset were scaled to 50×50 pixels and then randomly assigned to the training and test sets. Non-face images were added to the training set with a probability of 0.9053 to match the distribution of face images. Samples from both the

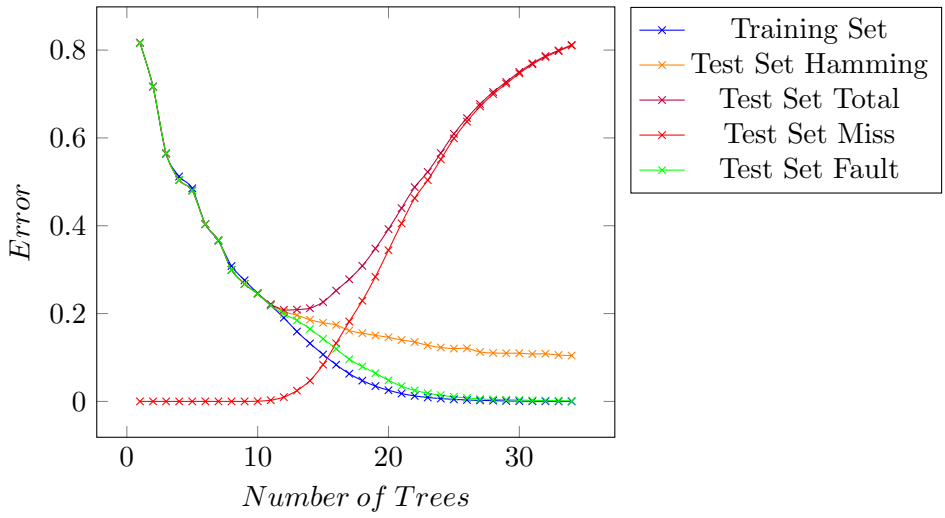


Fig. 3.3: Hamming Classification Performance for MNIST

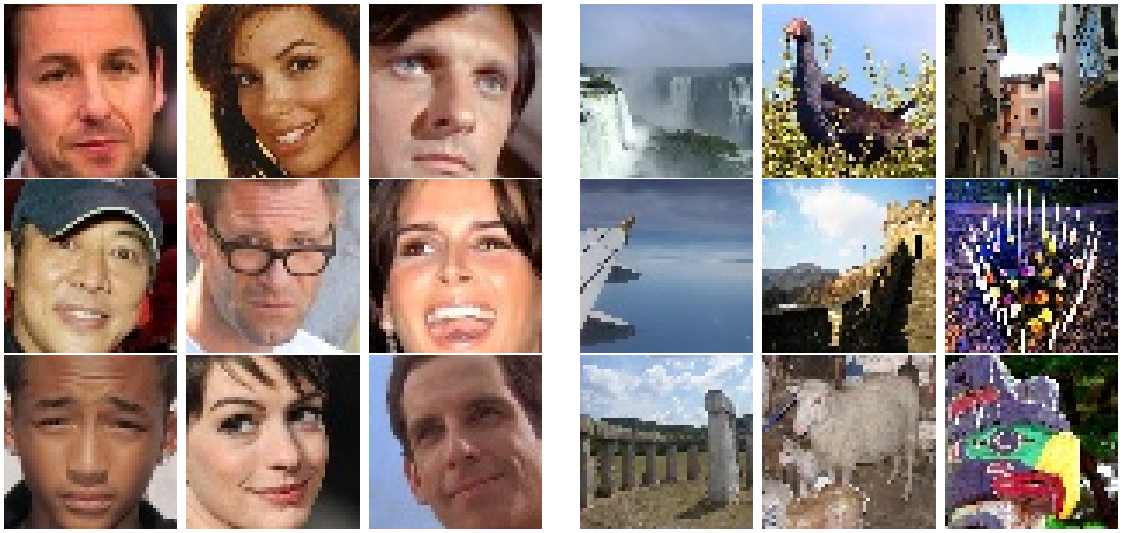
face and non-face images are included in Fig. 3.4.

Training until classification error was 0 took 53 trees. Fig. 3.5 shows the classification performance for the face detection dataset. Unlike the MNIST dataset the Hamming method keeps the error constant. This is to say the codec classifier performs the same with 12 trees as it would with 53 trees.

To help visualize trees the threshold and pixel of each tree is superimposed onto an image from the training set. If a tree operated on a pixel, then the color it operated on was given the value of the threshold while the other two colors were set to 0. This creates a colored pixel at the tree's location and with an intensity equal to that trees threshold. Fig. 3.6 shows the location and threshold of the first 7 pixels. Fig. 3.7 includes all trees.

Fig. 3.6 indicates that the first 7 trees' pixels are in proximity of the forehead, chin, nose, both cheeks, and both eyes. Furthermore, the trees are splitting on red in the forehead, cheeks, nose, and chin while the trees operating on pixels in the left and right eyes are splitting on blue and green respectively.

The training was also run on grayscale images. Each image in the training and test sets were converted to grayscale. Fig. 3.8 shows the classification error. 63 trees were necessary to drive the training error to 0.



(a) Face Image Samples (b) Non-face Image Samples

Fig. 3.4: Face Detection Dataset Image Samples

The resultant pixel location for the trained trees is shown in Fig. 3.9 and Fig. 3.10. The first 7 trees' pixels were again in the areas of the forehead, nose, chin, cheeks, and eyes.

To further help understand how the codec classifier was selecting trees mutual information images were developed. For every tree the mutual information is found for every pixel threshold combination. To generate the images the maximum mutual information value for each pixel is stored. Then the values are scaled to fill the range $0 - 255$ and displayed using a heat map. Brighter colors correspond to higher mutual information values. The first 8 images are included in Fig. 3.11. All 63 images are included in Appendix A.

The image associated with the first tree shows that the cheeks have a high mutual information value. Indicating that this would be a good operating pixel for the first tree. Subsequent images show the effect of previous trees. The second image has a dark spot on the left cheek where the first tree operates. Adding another tree with that pixel would contribute little to the ensemble's classification ability. Additionally, the addition of the first tree has reduced the mutual information value for several pixels in proximity to the first tree's pixel.

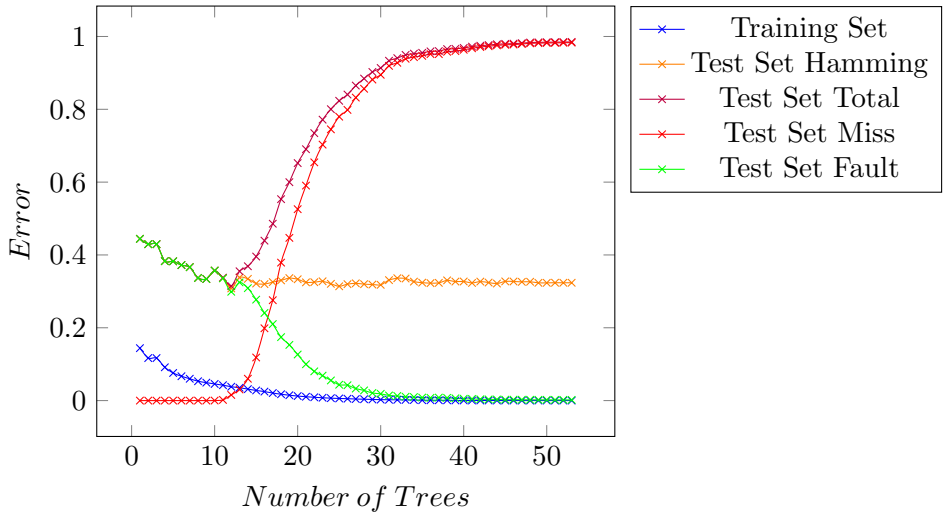


Fig. 3.5: Color Face Detection Classifications Performance

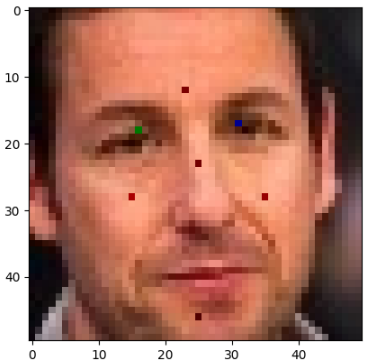


Fig. 3.6: First 7 Trees for Color Face Dataset

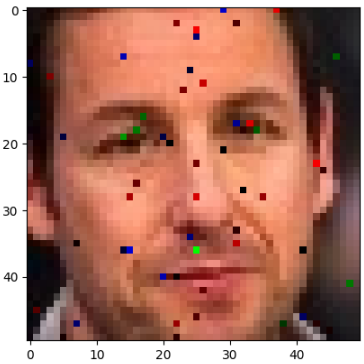


Fig. 3.7: All Trees for Color Face Dataset

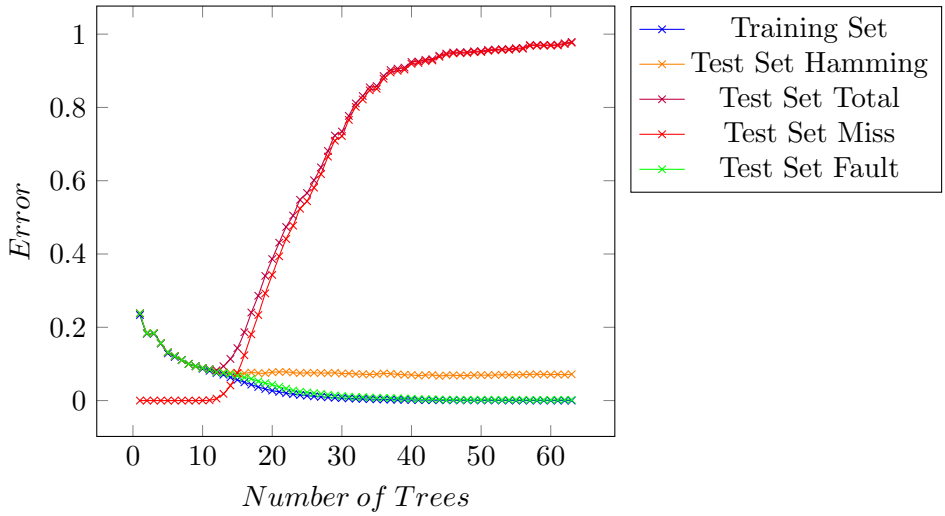


Fig. 3.8: Grayscale Face Detection Classifications Performance

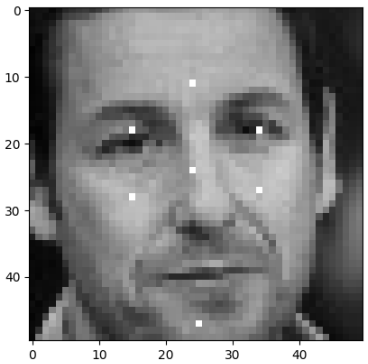


Fig. 3.9: First 7 Trees for Grayscale Dataset

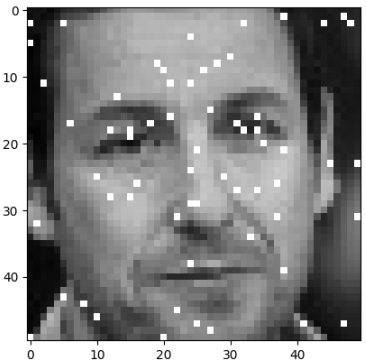
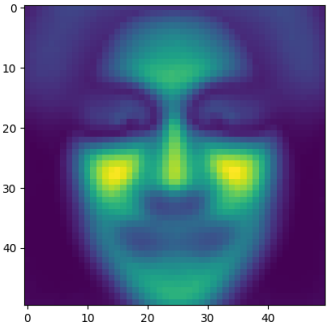
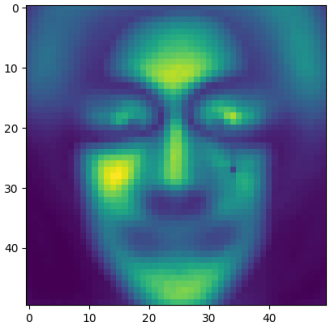


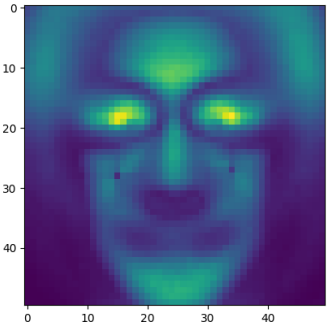
Fig. 3.10: All Trees for Grayscale Dataset



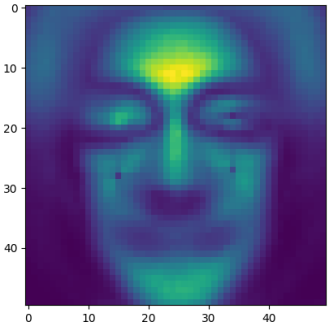
(a) Tree 1



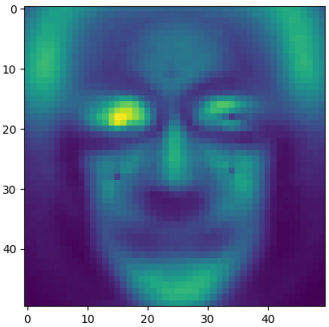
(b) Tree 2



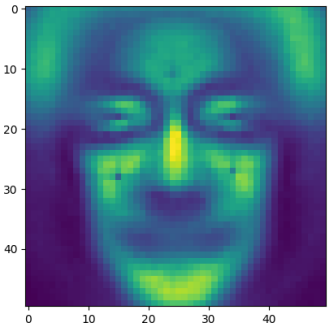
(c) Tree 3



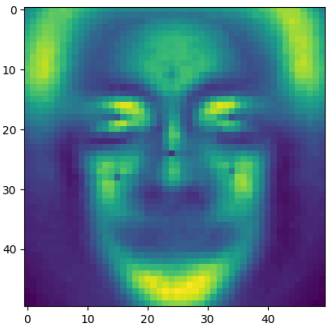
(d) Tree 4



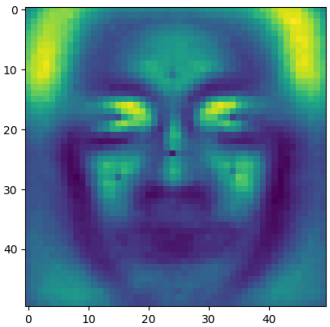
(e) Tree 5



(f) Tree 6



(g) Tree 7



(h) Tree 8

Fig. 3.11: Mutual Information Images for First 8 Trees

CHAPTER 4

Conclusions and Future Work

4.1 Conclusions

The results in this thesis show that the Codec Classifier is capable of classifying complex datasets with two or more classes. The reduced training complexity of the encoder using the ensemble dependency graph estimator of mutual information allows for fast local training or training on the edge. On MNIST the codec classifier with 34 trees and using the Hamming distance performs with an error of 10.44%. Adaboost with an equivalent number of trees has an error of 39.18%. While the Hamming distance decoding rule doesn't improve the error for the face dataset, the error for the grayscale data is still impressive at 7.19%. The pixels and thresholds of the first 7 trees align with what is intuitively characteristic of human faces. The mutual information images, which show the highest mutual information for each pixel, further confirm that the encoder in our method extracts informative features.

4.2 Future Work

Several changes to the encoder and decoder are possible. Some changes may improve the accuracy or efficiency of the classifier. It has been proposed that in the encoder a tree's output should be the result of all pixels weighted according the maximum mutual information of that pixel. So that for any tree t_i the output would be described by

$$t_i = u(-0.5 + \sum_{j=0}^{N} w_j[p_j < h_j]), \tag{4.1}$$

where t_i is the i^{th} tree, N is the number of pixels, p_j is the value of the j^{th} pixel, h_j is the threshold of the j^{th} pixel, w_j is the weight for the j^{th} pixel, and $u(\cdot)$ is the unit step function. Variants of this idea include taking a percentage of pixels with the highest mutual

information scores or taking all pixels whose mutual information falls within a range of the maximum mutual information.

Currently a codeword is decoded by comparing codewords from the training data using Hamming distance. For paralleled architectures like FPGAs and GPUs this process is very fast because it can be split among several processing units. For CPUs or processors with limited cores a linked list has been proposed for decoding. The depth of the linked list would be equal to the length of the codewords. Each node in the list would contain two references to other nodes. If a codeword is decoded that exist in the training set, then each bit would determine the path through the linked list until the last node is reached. This node contains class information. If a codeword is not contained in the training set, then at some point the path called for by a bit will not exist. In this case the codeword can take on the most common label of all children of that node. This information can be stored in the node to remove unnecessary searching. Fig. 4.1 demonstrates this approach when the codewords 000, 001, and 100 are used to create the list and 101 is decoded.

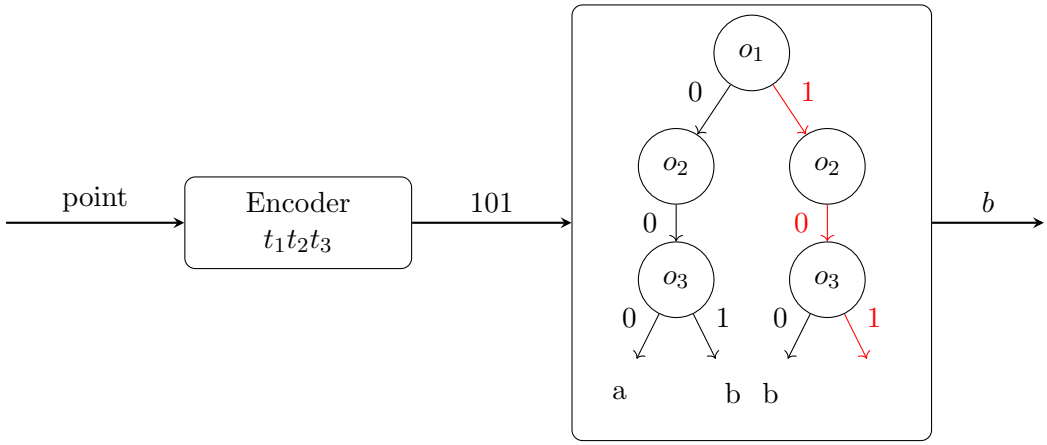


Fig. 4.1: Point Classification Using Linked Decoding

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] J. Treboux, D. Genoud, and R. Ingold, "Decision tree ensemble vs. n.n. deep learning: Efficiency comparison for a small image dataset," in *2018 International Workshop on Big Data and Information Security (IWBIS)*, 2018, pp. 25–30.

[3] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen, *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[4] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Computational Learning Theory*, P. Vitányi, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 23–37.

[5] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996. [Online]. Available: https://api.semanticscholar.org/CorpusID:47328136

[6] Y. Freund and R. E. Schapire, "A short introduction to boosting," in *Computer Science, Mathematics*, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:9621074

[7] M. Kawakita, M. Minami, S. Eguchi, and C. Lennert-Cody, "An introduction to the predictive technique adaboost with a comparison to generalized additive models," *Fisheries Research*, vol. 76, no. 3, pp. 328–343, 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016578360500216X

[8] T. M. Cover and J. A. Thomas, *Elements of Information Theory 2nd Edition (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, July 2006.

[9] M. Noshad, Y. Zeng, and A. O. Hero, "Scalable mutual information estimation using dependence graphs," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 2962–2966.

[10] Y.-I. Moon, B. Rajagopalan, and U. Lall, "Estimation of mutual information using kernel density estimators," *Phys. Rev. E*, vol. 52, pp. 2318–2321, Sep 1995. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.52.2318

[11] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical Review E*, vol. 69, no. 6, Jun. 2004. [Online]. Available: http://dx.doi.org/10.1103/PhysRevE.69.066138

[12] N. Kwak and C.-H. Choi, "Input feature selection by mutual information based on parzen window," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, pp. 1667– 1671, 01 2003.

[13] W. Gao, S. Kannan, S. Oh, and P. Viswanath, "Estimating mutual information for discrete-continuous mixtures," 2018.

APPENDICES

APPENDIX A

Mutual Information Images

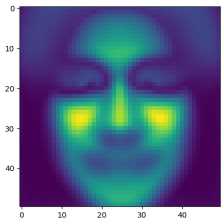


Fig. A.1: Mutual Information Image For Tree 1

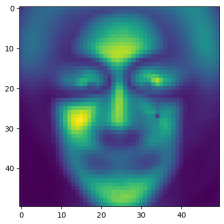


Fig. A.2: Mutual Information Image For Tree 2

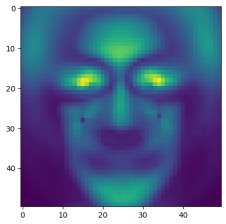


Fig. A.3: Mutual Information Image For Tree 3

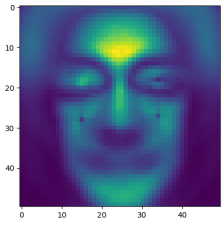


Fig. A.4: Mutual Information Image For Tree 4

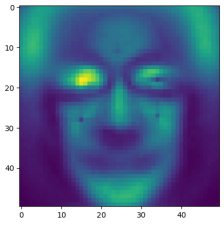


Fig. A.5: Mutual Information Image For Tree 5

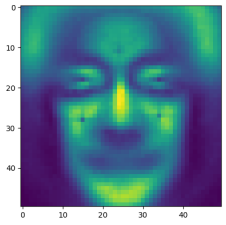


Fig. A.6: Mutual Information Image For Tree 6

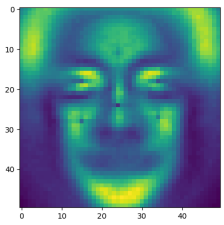


Fig. A.7: Mutual Information Image For Tree 7

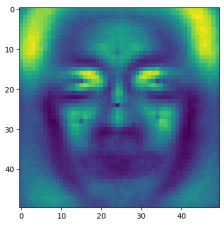


Fig. A.8: Mutual Information Image For Tree 8

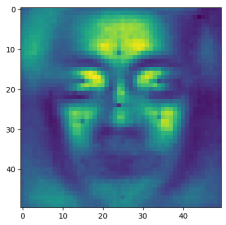


Fig. A.9: Mutual Information Image For Tree 9

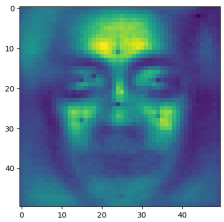


Fig. A.10: Mutual Information Image For Tree 10

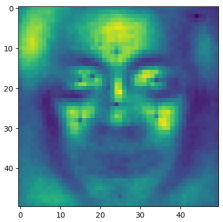


Fig. A.11: Mutual Information Image For Tree 11

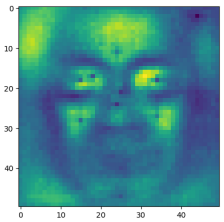


Fig. A.12: Mutual Information Image For Tree 12

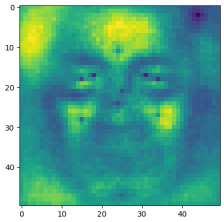


Fig. A.13: Mutual Information Image For Tree 13

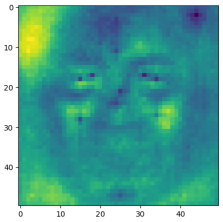


Fig. A.14: Mutual Information Image For Tree 14

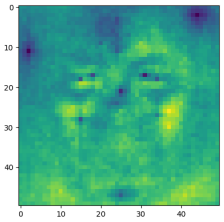


Fig. A.15: Mutual Information Image For Tree 15

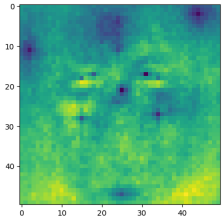


Fig. A.16: Mutual Information Image For Tree 16

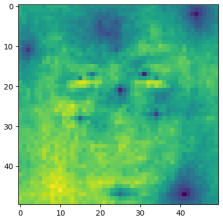


Fig. A.17: Mutual Information Image For Tree 17

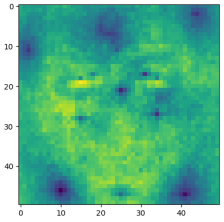


Fig. A.18: Mutual Information Image For Tree 18

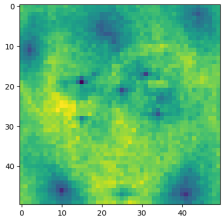


Fig. A.19: Mutual Information Image For Tree 19

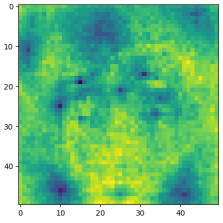


Fig. A.20: Mutual Information Image For Tree 20

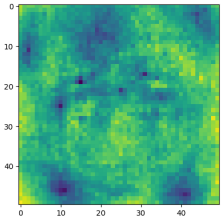


Fig. A.21: Mutual Information Image For Tree 21

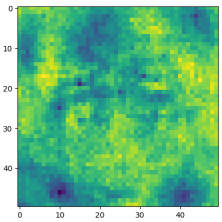


Fig. A.22: Mutual Information Image For Tree 22

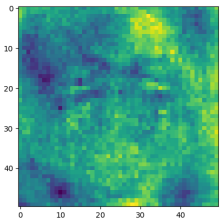


Fig. A.23: Mutual Information Image For Tree 23

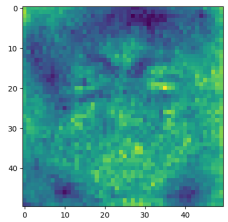


Fig. A.24: Mutual Information Image For Tree 24

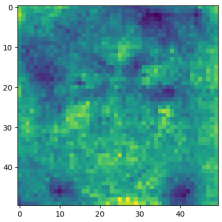


Fig. A.25: Mutual Information Image For Tree 25

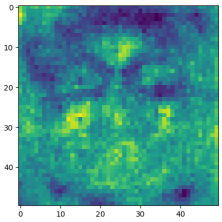


Fig. A.26: Mutual Information Image For Tree 26

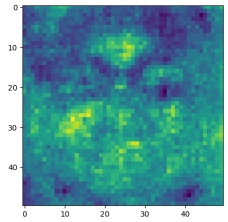


Fig. A.27: Mutual Information Image For Tree 27

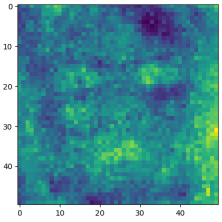


Fig. A.28: Mutual Information Image For Tree 28

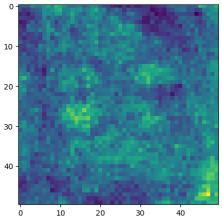


Fig. A.29: Mutual Information Image For Tree 29

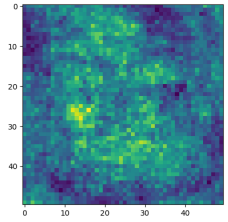


Fig. A.30: Mutual Information Image For Tree 30

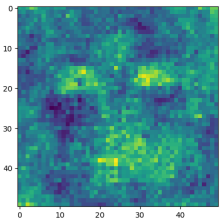


Fig. A.31: Mutual Information Image For Tree 31

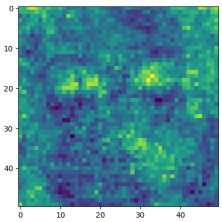


Fig. A.32: Mutual Information Image For Tree 32

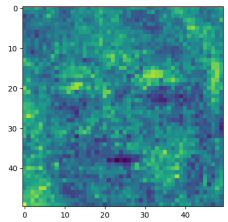


Fig. A.33: Mutual Information Image For Tree 33

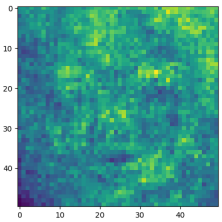


Fig. A.34: Mutual Information Image For Tree 34

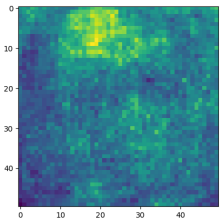


Fig. A.35: Mutual Information Image For Tree 35

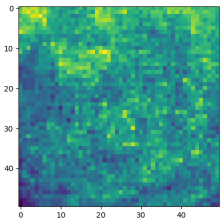


Fig. A.36: Mutual Information Image For Tree 36

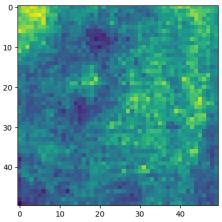


Fig. A.37: Mutual Information Image For Tree 37

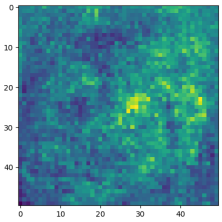


Fig. A.38: Mutual Information Image For Tree 38

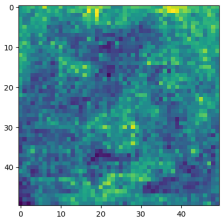


Fig. A.39: Mutual Information Image For Tree 39

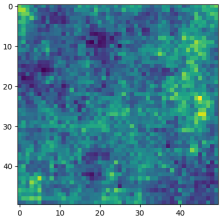


Fig. A.40: Mutual Information Image For Tree 40

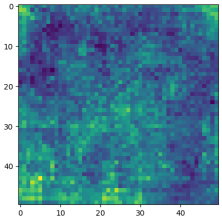


Fig. A.41: Mutual Information Image For Tree 41

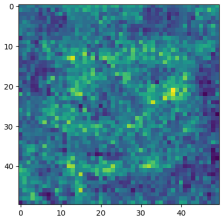


Fig. A.42: Mutual Information Image For Tree 42

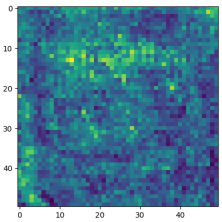


Fig. A.43: Mutual Information Image For Tree 43

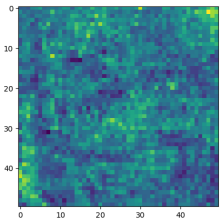


Fig. A.44: Mutual Information Image For Tree 44

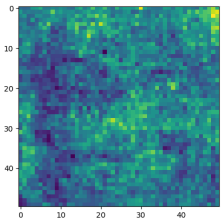


Fig. A.45: Mutual Information Image For Tree 45

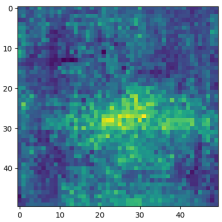


Fig. A.46: Mutual Information Image For Tree 46

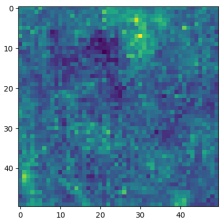


Fig. A.47: Mutual Information Image For Tree 47

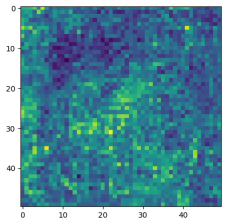


Fig. A.48: Mutual Information Image For Tree 48

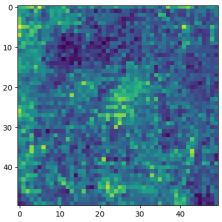


Fig. A.49: Mutual Information Image For Tree 49

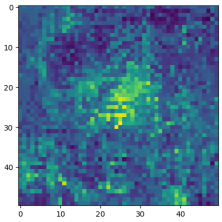


Fig. A.50: Mutual Information Image For Tree 50

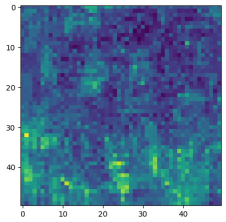


Fig. A.51: Mutual Information Image For Tree 51

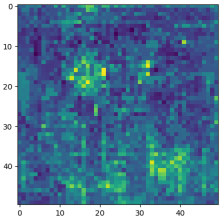


Fig. A.52: Mutual Information Image For Tree 52

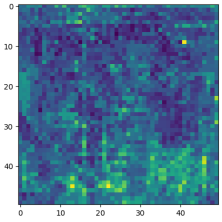


Fig. A.53: Mutual Information Image For Tree 53

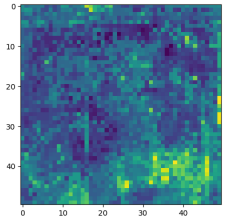


Fig. A.54: Mutual Information Image For Tree 54

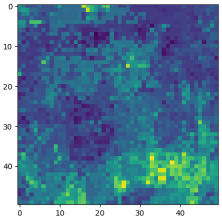


Fig. A.55: Mutual Information Image For Tree 55

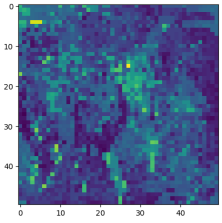


Fig. A.56: Mutual Information Image For Tree 56

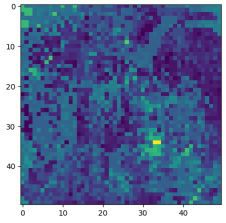


Fig. A.57: Mutual Information Image For Tree 57

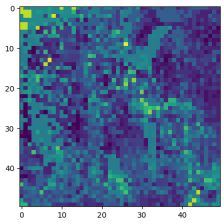


Fig. A.58: Mutual Information Image For Tree 58

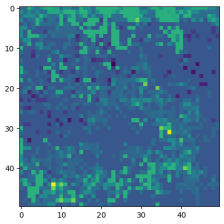


Fig. A.59: Mutual Information Image For Tree 59

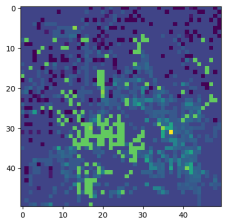


Fig. A.60: Mutual Information Image For Tree 60

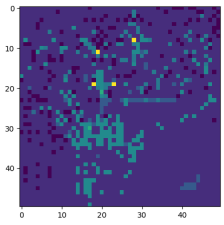


Fig. A.61: Mutual Information Image For Tree 61

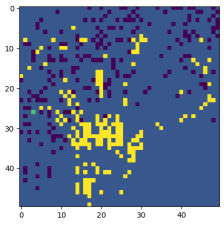


Fig. A.62: Mutual Information Image For Tree 62

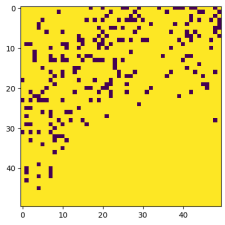


Fig. A.63: Mutual Information Image For Tree 63

CURRICULUM VITAE

Arle S. Beckwith

Published Journal Articles

- Residential water meters as edge computing nodes: Disaggregating end uses and creating actionable information at the edge, Attallah, N. A., Horsburgh, J. S., Beckwith, A. S., Tracy, R. J., *Sensors*, 2021.
- Impact of Data Temporal Resolution Quantifying Residential End Uses of Water, Bastidas Pacheco, C.J.; Horsburgh, J.S.; Beckwith, A.S., Jr., *Water*, 2022.