ENTROPY AS A CRITERION FOR VARIABLE REDUCTION IN CLUSTER DATA

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Christopher Wesley Olson

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department: Statistics
Program Name: Applied Statistics

April 2012

Fargo, North Dakota

North Dakota State University
Graduate School

Title

Using Entropy as a Criterion for Variable Reduction in Cluster Data

By

Christopher Wesley Olson

The Supervisory Committee certifies that this thesis complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Volodymyr Melnykov

Chair

Dr. Rhonda Magel

Dr. Seung Won Hyun

Dr. Edward Deckard

Approved:

5/2/2012 Dr. Rhonda Magel

Date Department Chair

1. ABSTRACT

Entropy is a measure of the randomness of a system state. This quantity gives us a measure of uncertainty that is associated with each particular observation belonging to a specific cluster. We examine this property and its potential use in analyzing high dimension datasets. Entropy proves most interesting in identifying possible dimensions that do not contribute meaningful classification to the clusters present. We can remove the dimension(s) found which are the least important and generalize this idea to a procedure. After identifying all the dimensions that should be eliminated from the dataset, we then compare its ability in recovering the true classification of the observations versus the estimated classification of the data. From the results obtained and shown in this paper, it is clear that entropy is a good candidate for a criterion in variable reduction.

TABLE OF CONTENTS

2. LIST OF TABLES

3. LIST OF FIGURES

4. INTRODUCTION

In many multivariate training datasets such as *Wine* [1], some of the dimensions measured are unimportant in the final model. These variables, often known simply as noise variables, do not contribute much if any useful information to the problem at hand and obscure the probabilistic nature of the other relevant dimensions. Variable reduction methods are often employed to help clean out this noise and find the important variables that remain. In this paper we introduce a new criterion to tackle this problem. Entropy will be used in reducing variables in a high dimensional dataset while testing its ability to do so. We consider the qualities of entropy within a clustering-based framework. The sections below explain the necessities in understanding what cluster analysis is, why it's important, and what it is used for, so that we have some basis to start from.

4.1 CLUSTER ANALYSIS

The complexity of this problem requires a brief explanation of its specific syntax and why cluster analysis is relevant in problems today. Cluster Analysis allows us to group similar observations together into what are called clusters. This is done by some measure defined within the framework of the type of clustering we are working in. These measures are known as criterion. The final output for the model identifies points that belong to different clusters, hopefully identifying the majority of these observations correctly. Groups of similar observations most commonly come in Gaussian densities or similar elliptical shapes. It can be a problem that the densities are non-spherical in nature and the algorithms used to determine the cluster groups will have trouble identifying the points correctly[2, 3]. This is known as misclassification and is covered in the next section. Within this large subject of clustering, we generally break it into three different net disciplines[4]: Partitioning Clustering, Hierarchical Clustering, and Model-Based Clustering.

Partition Clustering, or clustering that attempts to break the dataset into separate partitions, is a popular approach to cluster analysis. One particular algorithm that follows this train of thought is k-means. This is a procedure that given a number of clusters k, we

start with some initial partition of the dataset. We then try to improve the partitioning iteratively by moving objects or reassigning them step by step. The k-means approach works as follows - pick k distinct observations at random, and assume that they represent the cluster centers. Assign observations to clusters with the nearest cluster center. Recalculate the means for the given clusters with these new points and repeat this process until a stable solution is found. Partition-Based Clustering is great because it is easy to implement, and fast to calculate. However, initialization and distance measures often make it a poor estimate for non-spherical cluster groups that may exist [5].

Hierarchical Clustering works in a tree diagram fashion, akin to a nested folder structure. This type of clustering is based on a linkage rule that determines how we merge clusters based on some defined distance measure. At each level of the tree, or dendrogram, it specifies the number of clusters, k, we want to have. In essence, we cut the tree at a level k for how many clusters we want in the resulting dataset. Some typical linkages seen in literature include Single [measures distance between closest observations], Complete [measures distance between two distant observations], and Centroid [measures distance between cluster centers]. Hierarchical Clustering makes it easy to understand the resulting output, but it is a deterministic process that prevents re-evaluation of a single point after it is grouped in a cluster [4].

Model-Based Clustering is the method of implementation for the algorithm in this paper. As we explain further in the posterior probability section (section 5), the relative ease of using the Expectation-Maximization (EM) algorithm with pre-built packages in **R** [6] such as *mclust* makes simulation easily tackled. The groups of data in model-based clustering are defined by some distribution, multivariate normal or other common distributions that have nice features to work with. The problem is assumed such that the data come from a mixture of underlying probability distributions in which each part represents a separate cluster. Given the number of independent observations x_i, let $f_k(x_i|\theta_k)$ be the corresponding probability density function for the kth component of the model. Using a mixture likelihood approach [7], we maximize equation 1 to achieve estimate for Σ and μ matrices. In the

equation below, τ_k represents the probability of an observation belonging to a particular cluster k.

$$\mathcal{L}(\theta_1, ..., \theta_G; \tau_1, ..., \tau_G|\mathbf{x}) = \prod_{i=1}^{n} \sum_{k=1}^{G} \tau_k f_k(\mathbf{x}_i|\theta_k) \tag{1}$$

Under our simulation constraints, we are only concerned when the probability distribution of the components are the multivariate normal. This implies the parameter θ_k consists of the mean vector μ_k and covariance matrix Σ_k and the probability densities have the following familiar form:

$$f_k(\mathbf{x}_i|\mu_k, \Sigma_k) = (2\pi)^{-p/2}|\Sigma_k|^{-1/2} * \exp(-\frac{1}{2}(\mathbf{x}_i - \mu_k)^T \Sigma_k^{-1}(\mathbf{x}_i - \mu_k)) \tag{2}$$

After assigning some initial grouping of the points, the algorithm recomputes the estimate for the mean vector, and covariance matrix for each component [8]. We then use this step to calculate the needed criterion for the current step we are on, whether it be BIC, AIC, or Entropy as we are introducing in this paper. The previously computed parameters are now used to get better estimates and it repeats the process until we reach a convergence where the estimates don't change for some small epsilon. One of the big drawbacks of Model-Based Clustering is that it can be much more difficult to implement than either of the two previous methods but it will generally be more flexible for clusters present.

4.2 IDENTIFICATION AND MISCLASSIFICATION

Two of the more interesting parts of cluster analysis have to do with the correct classification of points and how to minimize the probability of misclassification of these points. If we develop a model that can do these two things well, it can then be used for a multitude of different applications as it becomes a way to group the data by certain characteristics. The noise variables seen in multivariate datasets often make it quite difficult to maintain a high percentage of identification whilst at the same time minimizing the probability of misclassification. In an illustrative example fully fleshed out in the algorithm

implementation section, we find that when the noise variables are removed from the simulated dataset, we can identify over 99% points belonging to their correct clusters. Compared to a paltry 74% without removing these extra dimensions, it is clear that the correct identification of points belonging to certain clusters can play a big part in determining how well your criterion can operate under the pressure.

4.3 ENTROPY

Shannon-Entropy, originally postulated in a paper on information theory in Bell Labs [9] by Claude Shannon is the quantity we are using for our criterion. This article explained how information entropy could be used as a measure for uncertainty in a message. We use it today for numerous problems, but it is most prevalent in information theory and thermodynamics. The formula for Gibbs-Entropy pertaining to thermodynamic systems is slightly different, but is comparable for what each quantity is measuring. As physicists are more concerned about changes in entropy instead of the quantity itself, there is no link between these two entropies besides the commonality in the concept of what they are measuring.

Entropy is traditionally thought of as a thermodynamics or information theory concept [10], but can be quite useful in a statistical framework as well. It measures the relative order of the system state, with a lower value implying a more well-ordered state and a larger value indicating a more chaotic dispersed state of the system. A typical example to explain this abstract concept involves a glass of ice in a room. As the ice melts, the order of the system (the contents of the glass) goes from a well-ordered state of ice cubes, to a much more chaotic state of water. The system starts out as a relatively rigid body of molecules, but as we add more heat energy to it, the water molecules begin to move about more thus making a much more random state. Shannon-Entropy is formally defined as the following below. When referring to entropy from this point onward, I will be using Shannon-Entropy

unless otherwise stated.

$$S = -\sum_{j=1}^{k}\sum_{i=1}^{n} \tau_{ij} * \log(\tau_{ij}) \tag{3}$$

It is a double sum over the observations n and number of clusters present k. For each individual point i, we take the posterior probability τ_{ij} of the point belonging to a specific cluster j and multiple it by the natural log of this probability. We then sum over all the points for this cluster and repeat the process by going through all clusters present. Generally speaking, there are no assumptions for the calculation initially as one can obtain the posterior probabilities from a non-parametric implementation [11]. In this work we emphasize the use of posterior probabilities obtained as a result of the EM algorithm in model-based clustering. This framework relies on a multivariate Gaussian finite mixture model, as described before. For this purpose, the **R** package *mclust* can be conveniently employed.

5. ALGORITHM IMPLEMENTATION

As reading code can get quite complicated quickly, an outline to the procedure is often helpful. A first pass of the code developed under this investigation is given below in two lists. The lists detail the steps taken for both forward and backward processes implemented in the code and follow similar approaches from other criterion-based algorithms [2]. After the forward and backward steps, a description of an entire problem example is given step by step. Part of the code developed for this paper is attached as an addendum and can also be requested from the author.

5.1 PSEUDO CODE

Backwards Selection Procedure Outline:

1. Given an initial set of k clusters, and a p-dimensional dataset find the posterior probabilities τ_{ij} for every point x_i by running the EM algorithm.

2. Calculate the initial entropy S_0 including all variables present. Use Equation 3 for the calculation of this statistic.

3. Take out each variable from the dataset, calculate what the entropy would be from a random sample with replacement with this new reduced dataset.

4. Repeat step 2 for a number n replications. Develop reference distribution of the entropy for each of the variables removed from the current model.

5. Make inference from the reference distribution and the initial entropy of the model at the current step. Drop the least significant variable from the list of insignificant variables.

5. Repeat steps 2-5 until there are no more rejections of insignificant variables.

The forward selection procedure is similar but different enough to demand it's own outline. Both of these procedures are akin to processes in regression [12] as they work in a similar fashion but these algorithms use entropy as a criterion instead of typical criterion-based statistics like AIC or BIC. The large difference between these two procedures occurs

in runtime performance, at least from empirical evidence. The forward selection appears to be much faster than the backwards selection process, but many factors can influence runtime performance during simulation such as the number of informative variables present. However, from the results shown later, both procedures work reasonably well and there are cases where one might be preferred over the other. This challenges the experimenter to choose the method appropriate for the problem, not because it just has performed well in the past.

Forwards Selection Procedure Outline:

1. Given an initial set of k clusters, and a p-dimensional dataset find the posterior probabilities τ_{ij} for every point x_i by running the EM algorithm.

2. Calculate the initial entropy S_0 for each of the individual variables. Pick the lowest entropy as the starting point and initial variable choice. Use Equation 3 for the calculation of these statistics.

3. Add each variable one at a time to the current dataset and calculate what the entropy would be from a random sample with replacement with this new dataset.

4. Repeat step 3 for a number n replications. Develop reference distribution of the entropy for each of the variables added to the current model.

5. Make inference from the reference distribution and the current initial entropy of the model at this step. Add the most significant variable if the entropy of the new dataset falls within the rejection region.

6. Repeat steps 2-5 until there are no more rejections of significant variables.

5.2 ILLUSTRATIVE EXAMPLE

For the following example, consider a dataset with three clusters and a sample size of 150, giving an average cluster size n_k of 50. The maximum overlap allowed between

these clusters is 0.05. This quantity, defined by $MaxOmega$ in my simulation runs, is the maximum allowed sum of misclassification probabilities among the clusters present [13]. The graphic below in figure 1 gives a clear visualization of the dataset. The shades of grey of the points represent the true identification of the points, showing which cluster the observations really belong. As we can clearly see, two of the dimensions are noisy and not contributing to the clustering at all. Only one dimension actually contains relevant information to the cluster behavior of the data. This example given is only worked out in the backwards selection method for sake of brevity, but the forward procedure is quite similar and also works well.

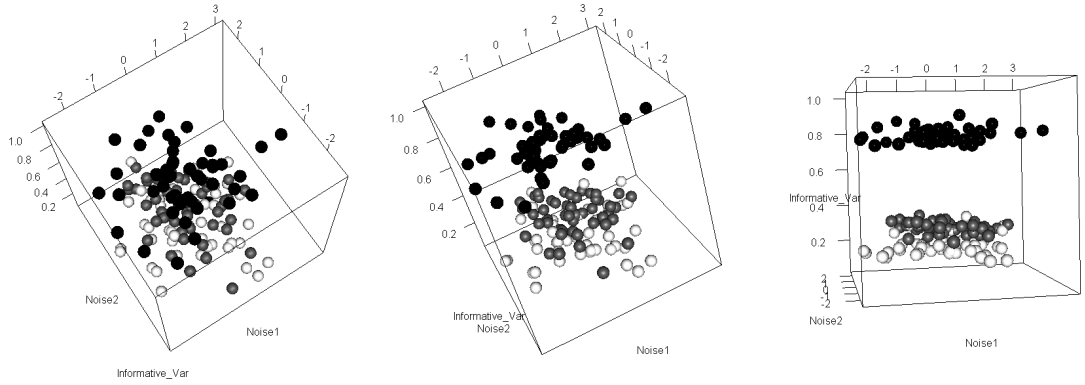

Figure 1: A three-dimensional plot of the initial dataset for our simulation.

After getting the initial entropy of the system with all variables present, we calculate the baseline of $S_0 = 35.21$. We now build up reference distributions for dropping any one of the variables from the system. In this case, since all three of the variables at the fifth percentile level are below our baseline measure of 35.21, we choose the lowest one, X2, in order to get the most order from removing this variable in the model. The fifth percentiles of each of the reference distributions are given in the table below. In the reference distribution graphic in figure 2, the dark line is the baseline measurement we calculated, S_0.

Since we dropped X_2 in the first iteration, our model now only contains dimensions X_1 and X_3. The entropy of the system has dropped from 35.21 to 1.036 from dropping the second dimension. This is a good indication we are on the right track for identifying

Variable	Entropy
S_0	35.21
X_1	28.69
X_2	0.215
X_3	0.547

Table 1: Percentiles and Initial Entropy - First Iteration.

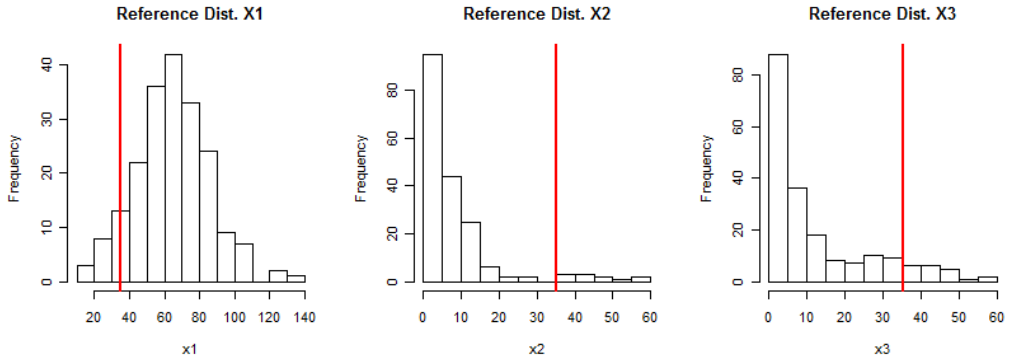

Figure 2: The reference distributions that are built by the algorithm in the first iteration.

important dimensions, as a lower entropy implies a much more well-ordered state. A new visualization of the system has been generated below in figure 3, and is slightly more clear where the clusters are in iteration two than in the initial iteration.

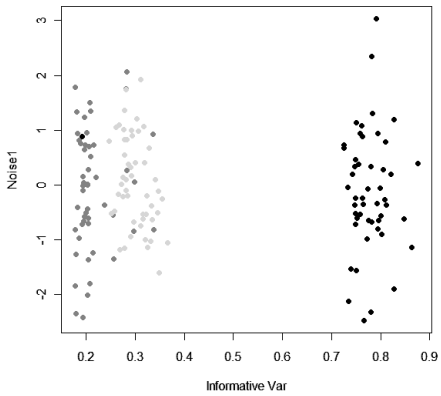

Figure 3: A visualization of our current dataset on the second iteration.

Once again, we recalculate the entropy of the system just as before, yielding $S_0 = 1.036$. We also build up reference distributions for dropping either of the remaining variables from the system. The fifth percentiles are given below for comparison just as before. Since X_3 will yield a lower entropy state than the current model if we drop it, it is the next variable that we eliminate in this procedure.

Variable	Entropy
S_0	1.036
X_1	49.32
X_3	0.127

Table 2: Percentiles and Initial Entropy - Second Iteration.

We reach our final iteration as we have ended up with only one variable left. In this case, our chosen variable X_1 yields a final entropy of $S_0 = 0.127$ which is a quite well-ordered state of the system. This procedure is great for identifying important variables, but it begs the question of what does this achieve for the experimenter and why do we go to such lengths to find these informative variables. The class proportion, the agreement between two classification schemes [14], will explain why this is such an important tool for us to explore. Initially, if we run a normal EM algorithm model-based clustering approach and get results for the observations belonging to specific clusters, we get a class proportion of 0.76. This means that about 76% of the observations will agree with their true identity or cluster. If we then run this same EM algorithm on the lone resulting informative variable, we get a class proportion of 0.993. This is a huge increase in information gleaned from the system. We can almost recover all of the correct classifications of points for the whole dataset. This is no easy feat when we have noise variables masking the true clustering and why variable selection is incredibly important for recovery of information from a dataset.

6. SIMULATION RESULTS

The graphics below in figures 4, 5 visually represent the bulk of simulation work done on checking the accuracy of entropy as a criterion with 1000 resamples done for each point. The tables that produced these graphs are attached as an appendix along with their corresponding confidence bounds for every point. There are two sets of images for each figure. The left picture has an easier clustering problem to solve on average, while the right image has a much more difficult problem to solve. The only difference between the left and right image is the cluster overlap, defined by the parameter *MaxOmega*. This parameter as previously mentioned, is the maximum allowed sum of misclassification probabilities among the clusters present. These figures are graphs of the proportion of times the informative variable and other variables stayed in the final model. These graphics are interesting to look at because we can tell the proportion of time the algorithm did not work at all in addition to seeing its drop off as we add more noisy variables to the problem.

The y-axis in these graphs reflects the relative proportion correct, while the x-axis defines the number of noisy variables present ranging from 1 to 8. The number of dimensions the algorithm has to tackle is the number of noise variables plus one, due to the informative variable as well. Another feature of these graphs are the dotted lines that are near the bottom of each image. These dotted lines represent the proportion of times where the algorithm simply spit back out the same variables it was given. If the lines were higher, it would imply something went wrong with the code, but thankfully all the graphs have a controllable proportion of failures.

After studying these images and looking at their corresponding tables, we can make a few observations about them for comparison. It is clear that both processes work well better in both sets of difficulties simulated. The efficiency in returning solutions that drop at least one non-informative variable indicate how well it is operating under different scenarios. Under the simulation efforts presented, there is no question that this criterion could be used in the cluster analysis efforts of today. The only major concern between the two different procedures however is the runtime efficiency. The forward selection method is about much

faster than the backwards selection method at least in this simulation study. It becomes a clear initial choice for any additional simulation efforts that might be undertaken in the future.

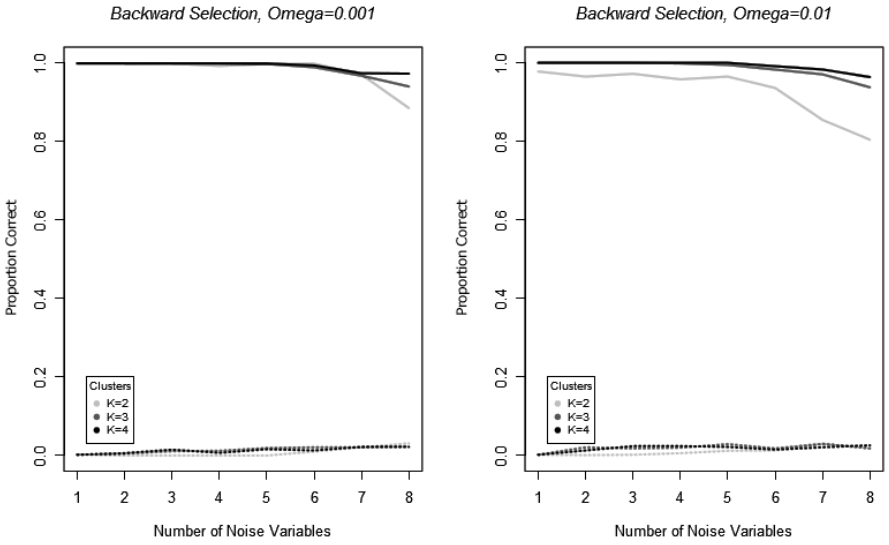

Figure 4: Containing Reduction - Backwards Selection.

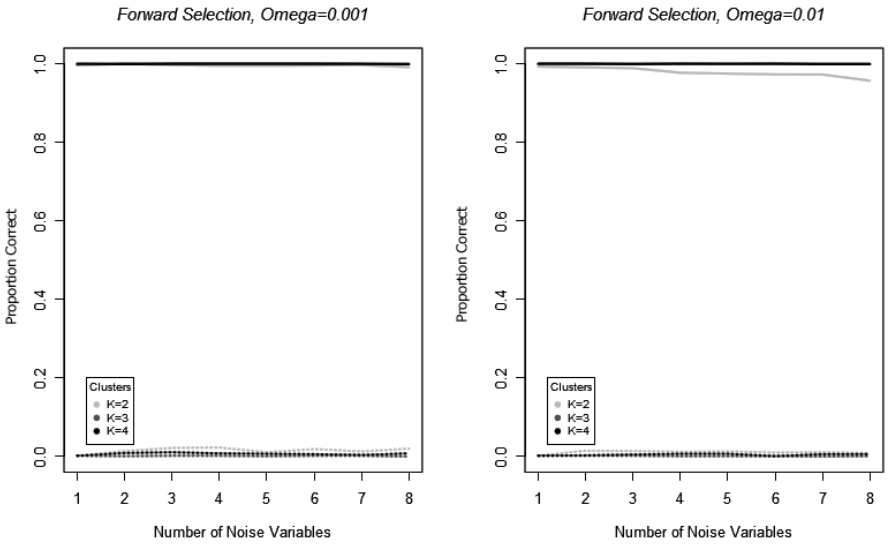

Figure 5: Containing Reduction - Forward Selection.

7. APPLICATION

Perhaps the real importance of the simulation results is how well the criterion works when used on real datasets. These datasets, like *Wine* and *Iris* may not be as simple to solve as some of the simulation efforts described above, so they provide a unique perspective on new criterion to be tested. Trying entropy on the *Wine* dataset [1] shows some promise. Using the forward selection method with the given 13 dimensional dataset, entropy was able to drop 6 variables. Initially, when all dimensions are present we get a class proportion agreement of 0.81. This means about 81% of the time, we will correctly place the observation to the right wine specified. After we drop the variables and run *mclust* again to get new classifications to compare with the originals, we get a much higher class proportion of 0.96. This boost in class proportion is a good sign that even when presented with data that may be messy, entropy can still be used to great effect. Even when not under the scope of strict simulation, it is able to increase the experimenter's ability in classifying observations correctly. This dataset is one of many that exist for training new criterion and we believe Entropy will be a good candidate to test against in similar datasets that exist.

8. OUTLINE FOR FURTHER RESEARCH

The posterior probabilities, τ_{ij} in the definition for entropy in equation 3 are calculated by the *mclust* [15] package in **R**. A common assumption to make simulation a much easier task to handle is to assume normality in our model. However, as mentioned previously, there is a way around this caveat. A non-parametric approach to estimating the probabilities is possible and provided below. This approach, while not implemented in this paper, is definitely a route for further research down the road. The time it would have taken to implement this in **R** was not feasible while simultaneously finishing the other code. The simulation presented in the paper was done entirely through the aid of *mclust* and *MixSim* as it makes the calculations and estimates relatively easy to obtain.

If we wish to estimate the posteriori, consider first estimating the density function $p(x)$ for some point in the system x. Construct some small range outside of x in every direction, defined as $R(x)$. We know we can estimate the probability mass function of $R(x)$ by $p(x) * v$, with v being the volume of the space of $R(x)$ [11]. We also know that $R(x)$ may be calculated from obtaining a large number of samples n from $p(x)$. Count the number of samples that fall within this small range, defined as k, with the simple relation k/n. We now have two relations for our small space around x and can solve the system to obtain the formula for estimating $p(x)$ [11].

$$p(x) = \frac{k}{nv} \tag{4}$$

The kth nearest neighbor approach to solving this for posteriori works in the following manner. Extend our region $R(x)$ until we get some number k neighbors as defined by the experimenter. Our formula then changes from above in equation 4 to the familiar Bayes' Rule for probabilities. This conclusion is called the k-nearest neighbor density estimate [11]. In the equation below, c_j refers to a particular cluster j that point x belongs to.

$$p(c_j|x) = \frac{p(c_j)p(x|c_j)}{p(x)} \tag{5}$$

Considering this equation, we can see that if we substitute n_j/n in for the estimate of $p(c_j)$, we simplify the above into the following expression. The estimate of $p(c_j|x)$ is simply the ratio of the number of points in cluster c_j and the number of points in our small window $R(x)$.

$$\begin{aligned} p(c_j|x) &= \frac{p(c_j)p(x|c_j)}{p(x)} \\ &= \frac{\frac{n_j}{n}\frac{k(x|c_j)}{n_j v}}{\frac{k(x)}{nv}} \\ &= \frac{k(x|c_j)}{k(x)} \end{aligned} \tag{6}$$

We now have an expression for calculating the posterior probabilities of any point belonging to a specific cluster. We can now use this equation and run similar simulations as above to observe any noticeable change in solution correctness.

9. CONCLUSIONS

Through simulation efforts above, we can conclude that entropy is a valid criterion for dimensional reduction. The efficiency of it dropping noisy variables is quite good and it maintains this efficiency as the number of noisy variables increases. Although the robustness of this equation was only tested through nine dimensions and one informative variable in this study, it shows promise for future use. In addition to implementing non parametric posteriori estimations, future work on this subject may include testing efficiencies of competing criterion such as AIC and BIC, as well as increasing the simulation to more dimensions and more informative variables present. We will be able to come to a definitive conclusion on how the criterion stack up against one another in a multitude of cluster scenarios.

10. BIBLIOGRAPHY

[1] A. Frank and A. Asuncion, “”UCI Machine Learning Repository”,” 2012.

[2] C. Maugis, G. Celeux, and M.-L. Martin-Magniette, “”Variable Selection in Model-Based Clustering: A General Variable Role Modeling”,” *Computational Statistics and Data Analysis*, vol. 53, no. 3872, 2009.

[3] S. Wang and J. Zhu, “”Variable Selection for Model-Based High-Dimensional Clustering and Its Application to Microarray Data”,” *Biometrics*, vol. 64, 2008.

[4] A. Rencher, *”Methods of Multivariate Analysis”*. John Wiley and Sons, Inc., 2nd ed., 2002.

[5] A. Peterson, A. Ghosh, and R. Maitra, “”A Systematic Evaluation of Different Methods for Initializing the K-means Clustering Algorithm”,” *IEEE Transactions on Knowledge and Data Engineering*, 2010.

[6] R. D. C. Team, *”R: A Language and Environment for Statistical Computing”*. R Foundation for Statistical Computing, 2012.

[7] C. Fraley and A. Raftery, *”How many Clusters? Which Clustering Method? Answers via Model-Based Cluster Analysis”*. Department of Statistics, Univeristy of Washington, 2006.

[8] A. Rafery and N. Dean, “”Variable Selection for Model-Based Clustering”,” *Journal of the American Statistical Association*, vol. 101, no. 473, 2006.

[9] C. Shannon, “”A Mathematical Theory of Communication”,” *The Bell System Technical Journal*, vol. 27, 1948.

[10] Gyftopoulos, *”Thermodynamics. Foundations and Applications”*. Dover, 3rd ed., 2010.

[11] H. Li, K. Zhang, and T. Jiang, “”Minimum Entropy Clustering and Applications to Gene Expression Analysis”,” in *In Proceedings of IEEE Computational Systems Bioinformatics Conference*, pp. 142–151, 2004.

[12] B. Abraham and J. Ledolter, *"Introduction to Regression Modeling"*. Thomson Brooks/Cole, 1st ed., 2006.

[13] R. Maitra and V. Melnykov, ""MixSim: an R Package for Simulating Data from Finite Gaussian Mixtures to Study Performance of Clustering Algorithms"," *Journal of Statistical Software*, 2012.

[14] M. Meila, ""Comparing Clusterings - An Information Based Distance"," *Jounal of Multivariate Analysis*, vol. 98, 2007.

[15] C. Fraley and A. Raftery, *"Model-Based Clustering / Normal Mixture Modeling"*. R Foundation for Statistical Computing, 2007.

A. TABLES FOR SIMULATION

1 NV	2 NVs	3 NVs	4 NVs	MaxOmega	Clusters
$.9960 \pm .0039$	$1.000 \pm --$	$.9969 \pm .0034$	$.9949 \pm .0044$	0.001	2
$.9990 \pm .0020$	$1.000 \pm --$	$1.000 \pm --$	$1.000 \pm --$	0.001	3
$1.000 \pm --$	$.9990 \pm .0020$	$1.000 \pm --$	$1.000 \pm --$	0.001	4
$.9930 \pm .0052$	$.9909 \pm .0059$	$.9888 \pm .0065$	$.9774 \pm .0092$	0.01	2
$1.000 \pm --$	$1.000 \pm --$	$1.000 \pm --$	$1.000 \pm --$	0.01	3
$1.000 \pm --$	$1.000 \pm --$	$.9990 \pm .0020$	$1.000 \pm --$	0.01	4
5 NV	6 NVs	7 NVs	8 NVs	MaxOmega	Clusters
$.9949 \pm .0044$	$.9958 \pm .0040$	$.9969 \pm .0035$	$.9915 \pm .0057$	0.001	2
$1.000 \pm --$	$1.000 \pm --$	$0.999 \pm .0020$	$1.000 \pm --$	0.001	3
$1.000 \pm --$	$1.000 \pm --$	$1.000 \pm --$	$0.990 \pm .0020$	0.001	4
$.9752 \pm .0096$	$.9731 \pm .0100$	$.9727 \pm .0101$	$.9571 \pm .0126$	0.01	2
$.9990 \pm .0020$	$1.000 \pm --$	$1.000 \pm --$	$1.000 \pm --$	0.01	3
$1.000 \pm --$	$1.000 \pm --$	$.9990 \pm .0020$	$.9990 \pm .0020$	0.01	4

Table 3: Table of calculated probabilities and variability for forward selection.

1 NV	2 NVs	3 NVs	4 NVs	MaxOmega	Clusters
$.9960 \pm .0039$	$.9960 \pm .0039$	$.9980 \pm .0028$	$.9920 \pm .0055$	0.001	2
$.9990 \pm .0020$	$.9988 \pm .0021$	$.9987 \pm .0022$	$.9986 \pm .0023$	0.001	3
$.9989 \pm .0021$	$.9987 \pm .0022$	$.9985 \pm .0024$	$.9983 \pm .0025$	0.001	4
$.9780 \pm .0091$	$.9650 \pm .0114$	$.9720 \pm .0102$	$.9580 \pm .0124$	0.01	2
$1.000 \pm --$	$1.000 \pm --$	$1.000 \pm --$	$.9987 \pm .0022$	0.01	3
$1.000 \pm --$	$1.000 \pm --$	$1.000 \pm --$	$1.000 \pm --$	0.01	4
5 NV	6 NVs	7 NVs	8 NVs	MaxOmega	Clusters
$.9970 \pm .0034$	$.9980 \pm .0028$	$.9700 \pm .0106$	$.8846 \pm .0198$	0.001	2
$.9971 \pm .0033$	$.9887 \pm .0066$	$.9671 \pm .0110$	$.9396 \pm .0148$	0.001	3
$.9984 \pm .0025$	$.9934 \pm .0050$	$.9733 \pm .0100$	$.9724 \pm .0102$	0.001	4
$.9650 \pm .0114$	$.9359 \pm .0152$	$.8546 \pm .0219$	$.8042 \pm .0246$	0.01	2
$.9948 \pm .0045$	$.9828 \pm .0081$	$.9706 \pm .0105$	$.9377 \pm .0150$	0.01	3
$1.000 \pm --$	$.9915 \pm .0057$	$.9830 \pm .0080$	$.9637 \pm .0116$	0.01	4

Table 4: Table of calculated probabilities and variability for backward selection.

B. BACKWARD ELIMINATION R CODE

```
require(mclust)
require(MixSim)
require(doSMP)


# function that returns the entropy of the passed dataset/clusters
entropy <- function(dataset,clusters)
{
  # find the entropy of the system of the current dataset
  total <- ncol(dataset)
  cluster <- NA
  if(dim(dataset)[2] > 1){
    if(inherits(try(Mclust(dataset,G=clusters, modelNames="VVV"),
    silent=TRUE), what = "try-error") == FALSE){
      cluster <- Mclust(dataset,G=clusters, modelNames="VVV")
    }
  }else{
    if(inherits(try(Mclust(dataset,G=clusters, modelNames="V"),
    silent=TRUE), what = "try-error") == FALSE){
     cluster <- Mclust(dataset,G=clusters, modelNames="V")
  }}
  # drop observations that cause errors in entropy calculation
  if(is.na(cluster) == FALSE){
    x <- cluster$z[!cluster$z==0]
    entropy <- (-1)*sum(x*log(x))
    return(entropy)
  }else{
    return(NA)
  }
}



model_selection <- function(dataset,clusters,resamples,alpha)
{
  # Create column names if they don't exist
  j <- NULL
  if(is.null(colnames(dataset))){
    for (i in 1:ncol(dataset)){
      columnname <- paste("x",i, sep="")
      if(length(j) < 1){
        j <- columnname
```

```
      }else{
      j <- c(j,columnname)
    }
  }
}
colnames(dataset) <- j


done <- 0
while(done == 0)
{
stringback <- colnames(dataset)
samplesize <- nrow(dataset)
options(warn=-1)
entropy_initial <- entropy(dataset,clusters)

if(is.na(entropy_initial))
{
  done == 1
  cat("mclust failed to initialize model (program halted)", "\n")
  return("ERROR")
}

# Print out current model
cat("Current Model... ")
for(i in 1:length(colnames(dataset)))
{
  cat(colnames(dataset)[i], " ")
}
cat(" Entropy:",round(entropy_initial,5))
cat("\n")

# Get entropy for a single dimension dropped in the current whole model and
# resample it
entropy_reduction <- matrix(NA,ncol=resamples,nrow=ncol(dataset))
rownames(entropy_reduction) <- colnames(dataset)
new_dataset <- array(NA,dim=c(samplesize,ncol(dataset)-1,
 ncol(dataset),resamples))

for(i in 1:ncol(dataset))
{
  new_dataset[,,i,] <- dataset[,-i]
}

sampled_data <- apply(new_dataset,c(2,3,4),
```

```
  function(x) sample(x,samplesize,replace=TRUE,prob=NULL))
  # Get and then compare entropies from the resamples
  entropy_reduction <- apply(sampled_data,c(3,4),function(x) entropy(x,clusters))
  entropy_comparison <- apply(entropy_reduction,1,
  function(x) quantile(x, probs=c(alpha),na.rm = TRUE)[1])
  names(entropy_comparison) <- stringback

  if(any(entropy_comparison <= entropy_initial))
  {
    inds <- which(entropy_comparison == min(entropy_comparison))
    dataset <- dataset[,-inds]
    done = 0
    if(is.matrix(dataset)==FALSE)
    {
      done = 1
      indskeep <- which(entropy_comparison == max(entropy_comparison),
        arr.ind=TRUE)
      cat("Final Model..... ",names(indskeep),
      " Entropy:",round(min(entropy_comparison),5),"\n")
      return(names(indskeep))
    }
  }

  if(all(entropy_comparison > entropy_initial))
  {
    cat("Final Model..... ")
    for(i in 1:length(colnames(dataset)))
    {
      cat(colnames(dataset)[i], " ")
    }
    cat(" Entropy:",round(entropy_initial,5))
    cat("\n")
    done = 1
    return(colnames(dataset))
  }
  }
}

### Function testing below here...

rmSessions(all.names=TRUE)
setwd("C:/Users/Chris/Desktop/Documents/Classes/NDSU/Thesis/Simulation")
w <- startWorkers(workerCount = 6, FORCE=TRUE)
registerDoSMP(w)
```

```
for(k in 4:4){
for(q in 8:8){
foreach(i = 1:420, .packages=c("mclust","MixSim")) %dopar%{
# dimensions / components for simulation
p <- 1
K <- k
size <- K*50
q <- q
omega <- 0.001

M <- MixSim(MaxOmega=omega, K=K, p=p)
D <- simdataset(n = size, Pi = M$Pi, Mu = M$Mu, S = M$S)
class <- as.matrix(D$X,ncol=p)
generation <- rnorm(size*q)
x <- matrix(generation,ncol=q,nrow=size)
test <- cbind(class,x)
z <- model_selection(test,K,100,0.05)

write(z,file=paste("n",size,"omega",omega,"K",K,"p",p,"q",q,".txt",sep=""),
ncolumns=q+p, append=TRUE)
}
}
}

stopWorkers(w)
```

C. FORWARD SELECTION R CODE

```
require(mclust)
require(MixSim)
require(doSMP)


# function that returns the entropy of the passed dataset/clusters
entropy <- function(dataset,clusters)
{
  # find the entropy of the system of the current dataset
  total <- ncol(dataset)
  cluster <- NA
  if(dim(dataset)[2] > 1){
    if(inherits(try(Mclust(dataset,G=clusters, modelNames="VVV"),
     silent=TRUE), what = "try-error") == FALSE){
      cluster <- Mclust(dataset,G=clusters, modelNames="VVV")
    }
  }else{
    if(inherits(try(Mclust(dataset,G=clusters, modelNames="V"),
     silent=TRUE), what = "try-error") == FALSE){
     cluster <- Mclust(dataset,G=clusters, modelNames="V")
  }}
  # drop observations that cause errors in entropy calculation
  if(is.na(cluster) == FALSE){
    x <- cluster$z[!cluster$z==0]
    entropy <- (-1)*sum(x*log(x))
    return(entropy)
  }else{
    return(NA)
  }
}


model_selection <- function(dataset,clusters,resamples,alpha)
{
  # Create column names if they don't exist
  j <- NULL
  if(is.null(colnames(dataset))){
    for (i in 1:ncol(dataset)){
      columnname <- paste("x",i, sep="")
      if(length(j) < 1){
        j <- columnname
        }else{
```

```
      j <- c(j,columnname)
    }
  }
}
colnames(dataset) <- j

dataset_initial <- dataset

# find lowest entropy variable to start
entropy_initial_model <- matrix(NA,ncol=ncol(dataset),nrow=1)
colnames(entropy_initial_model) <- colnames(dataset)
options(warn=-1)

for(i in 1:ncol(dataset))
{
  entropy_initial_model[1,i] <- entropy(as.matrix(dataset[,i]),
   clusters=clusters)
}

inds_initial <- which(entropy_initial_model == min(entropy_initial_model))
dataset <- as.matrix(dataset[,-inds_initial])
keeps <- colnames(entropy_initial_model)[inds_initial]

inds <- inds_initial
samplesize <- nrow(dataset)
done <- 0
while(done == 0)
{
  stringback <- colnames(dataset)

  options(warn=-1)
  entropy_initial <- entropy(as.matrix(dataset_initial[,inds]),clusters)

if(is.na(entropy_initial))
{
  done == 1
  cat("mclust failed to initialize model (program halted)", "\n")
  return("ERROR")
}

# Print out current model
cat("Current Model... ")
for(i in 1:length(keeps))
{
  cat(keeps[i], " ")
```

```
}
cat(" Entropy:",round(entropy_initial,5))
cat("\n")

# Get entropy for a single dimension added in
# the current whole model and resample it
entropy_reduction <- matrix(NA,ncol=resamples,nrow=ncol(dataset))
rownames(entropy_reduction) <- colnames(dataset)
new_dataset <- array(NA,dim=c(samplesize,1,ncol(dataset),resamples))

for(i in 1:ncol(dataset))
{
  new_dataset[,,i,] <- dataset[,i]
}

sampled_data <- apply(new_dataset,c(2,3,4)
 ,function(x) sample(x,samplesize,replace=TRUE,prob=NULL))
entropy_reduction <- apply(sampled_data,c(3,4),function(x) entropy(x,clusters))
entropy_comparison <- apply(entropy_reduction,1,
 function(x) quantile(x, probs=c(alpha),na.rm = TRUE)[1])
names(entropy_comparison) <- stringback

if(any(entropy_comparison <= entropy_initial))
{
  if(dim(dataset)[2]==1)
  {
    done = 1
    keeps <- c(names(which(entropy_comparison == min(entropy_comparison))),keeps)
    cat("Final Model..... ")
    for(i in 1:length(keeps))
    {
      cat(keeps[i], " ")
    }
    cat("\n")
    return(keeps)
  }

  inds <- which(entropy_comparison == min(entropy_comparison))
  dataset <- as.matrix(dataset[,-inds])
  keeps <- c(keeps,names(inds))
  done = 0
}

if(all(entropy_comparison > entropy_initial))
{
```

```
    cat("Final Model... ")
    for(i in 1:length(keeps))
    {
      cat(keeps[i], " ")
    }
    cat(" Entropy:",round(entropy_initial,5))
    cat("\n")
    done = 1
    return(keeps)
  }
  }
}

rmSessions(all.names=TRUE)
setwd("C:/Users/Chris/Desktop/Documents/Classes/NDSU/Thesis/Simulation")
w <- startWorkers(workerCount = 6, FORCE=TRUE)
registerDoSMP(w)


for(k in 4:4){
for(q in 7:8){
foreach(i = 1:1000, .packages=c("mclust","MixSim")) %dopar%{
# dimensions / components for simulation
p <- 1
K <- k
size <- K*50
q <- q
omega <- 0.001

M <- MixSim(MaxOmega=omega, K=K, p=p)
D <- simdataset(n = size, Pi = M$Pi, Mu = M$Mu, S = M$S)
class <- as.matrix(D$X,ncol=p)
generation <- rnorm(size*q)
x <- matrix(generation,ncol=q,nrow=size)
test <- cbind(class,x)
z <- model_selection(test,K,100,0.05)

write(z,file=paste("n",size,"omega",omega,"K",K,"p",p,"q",q,"
.txt",sep=""),
ncolumns=q+p, append=TRUE)
}
}
}

stopWorkers(w)
```