Performance Comparison of ADRS and PCA as a Preprocessor to ANN for Data Mining

Nicholas Navaroli[1] David Turner[1] Arturo I Concepcion[1] Robert S Lynch[2]

[1]*Department of Computer Science and Engineering*
California State University, San Bernardino
{navaroln, dturner, concep} @csusb.edu

[2]*Naval Undersea Warfare Center*
Newport, RI
LynchRS@Npt.NUWC.Navy.Mil

Abstract

In this paper we compared the performance of the Automatic Data Reduction System (ADRS) and principal component analysis (PCA) as a preprocessor to artificial neural networks (ANN). ADRS is based on a Bayesian probabilistic classifier that is used with a quantization process that results in a simplification of the feature space, including elimination of irrelevant features. ADRS has the advantage of retaining the original names of the features even though the feature space has been modified. Thus, results are easier to interpret than those of PCA and ANN, which transform the feature space in a way that obscures the original meanings of the features. The comparison showed that ADRS performs better than PCA as a preprocessor to ANN when data mining the datasets of the UCI Machine Learning Repository.

1. Introduction

The Automatic Data Reduction System (ADRS) is a Java implementation of the Bayesian Data Reduction Algorithm (BDRA), which was developed by Robert S. Lynch and Peter K. Willett [1]. The BDRA is a probabilistic classifier that reduces irrelevant features from a set of training data for each class to produce better classification performance than other classifiers. BDRA through a quantization process, data is self-adjusted to produce the best results. It was also shown that BDRA can also be used for processing missing features in the data.

BDRA requires that all data be discrete, and so all continuous valued data is discretized using a method of percentiles. Once all features are discretized, BDRA proceeds to reduce those features automatically based on the probability of error conditioned on the training data and finds the quantization of features that produce the best classification performance.

Originally BDRA was implemented in Matlab and this imposes some restrictions on performance and limitations on which platforms BDRA can be executed. ADRS can be easily integrated with existing information processing systems and package as a Web application service in the future.

In this paper, we compared the performance of ADRS and PCA (principal component analysis) as preprocessors for artificial neural networks (ANNs). ANN is a common method used for data mining purposes, and PCA is the usual preprocessor used in connection with ANN. PCA is used to reduce the dimensionality of the feature space prior to training the ANN, which may improve the overall performance of ANN. ADRS can be used as an alternative to PCA because it can reduce the dimensionality of the dataset by eliminating irrelevant ones. ADRS retains the original names of the features, whereas PCA merges features into unrecognizable labels.

2. ADRS Parameters

There are three operational modes of ADRS: evaluation, training, and classify. In the evaluation mode, ADRS generates a report that describes how well the system classifies the test data given specific parameters that define the operation of the BDRA. The program inputs are the training datasets and the ADRS parameters. The program output is a report containing the estimated classification error and a feature analysis that describes how BDRA quantized each feature and which features BDRA eliminated. In the training mode, ADRS generates a classifier. The program inputs are the training datasets and the BDRA parameters. The program output is a classifier file. In the classify mode, ADRS uses the classifier file to classify observations. The program inputs are the unclassified observations and the classifier file. The program output is a classification report. This is shown in Figure 1 for the use of ADRS.

The ADRS parameters are:

- *Continuous Bins*: <positive integer>
- *Missing-Value Approach*: pseudo-bin | mean-field
- *Search Direction*: forward | backward
- *Search Technique*: sequential | winding
- *Hold-Out-Size*: <positive integer>

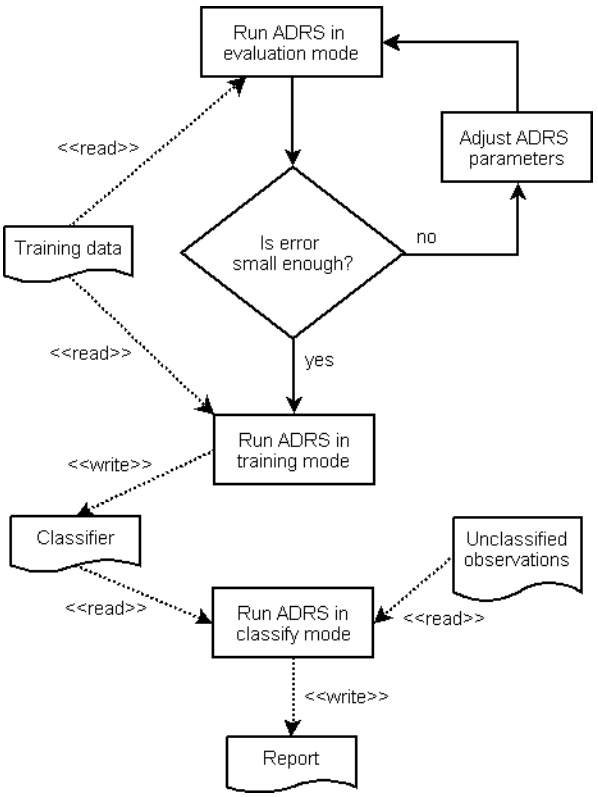

Figure 1. ADRS Usage

The continuous bins parameter specifies the number of bins used for categorizing the discretized values for those features that have continuous values. The missing value approach parameter is used to select one of two approaches to dealing with missing feature values in the training data. In the pseudo-bin approach, missing-value bins are created for features with missing values. In the mean-field approach, missing values are probabilistically distributed across multiple bins, so that a missing value contributes fractional amounts to several bins.

The search direction parameter is used to specify the starting point in the search through the solution space. In the forward search direction, the algorithm starts with a fully reduced classifier in which the bins of each feature are merged into a single bin. In the backward search direction, the algorithm starts in a fully expanded classifier in which no bins are merged. The search technique parameter is set to either sequential or winding.

To understand the influence of the search technique parameter, it should be understood that the BDRA algorithm performs a greedy search through space of all possible classification tables, and that at each step in the algorithm, it finds the next best re-composition of bins within a feature (as measured by an internal estimate of classification error). When the search technique is sequential, the algorithm will merge bins in the case of a forward search, or will split bins in the case of a backward search. When the search technique is winding, the algorithm will either perform a merge operation or a split operation depending on which results in a smaller error.

The hold-out size parameter determines how many observations from the training data are withheld and used for testing the predictive ability of the classifier resulting from the training. When the hold-out size is greater than 1, ADRS randomly selects that many observations to withhold. When the hold-out size is set to 1, ADRS will perform n evaluations in which each training observation is held out once, where n is the total number of observations. The total number of false predictions divided by n is taken as an estimate of the expected error in using the BDRA with the population from which the data set comes. The process for a holdout size of 1 is illustrated by the following algorithm:

```
errors = 0
for each ob in observations
  obs = observations - ob
  classifier = ADRS(obs)
  class = classifier.classify(ob)
  if ob.class != class
    errors = errors + 1
print errors/number_of_observations
```

3. PCA Parameters

PCA is a model that is used for many purposes, such as image recognition and finding patterns in multi-dimensional data [2]. First, each feature in the dataset is subtracted by that feature's mean value. The mean for each feature will then become 0. The covariance of each combination of features in the dataset is then calculated and stored into a covariance matrix. A dataset that contains A features will have a covariance matrix, C, with the following characteristics:

- Dimensions: A x A
- C(X,Y) → covariance of features X and Y

PCA then calculates all Eigenvectors of the matrix, and arranges them in ascending order according to their corresponding Eigenvalues. Each Eigenvector is represented as a 1 x A matrix. The Eigenvalues are normalized in such a way that the sum of all Eigenvalues will become 1 (each Eigenvalue is divided by the sum of all Eigenvalues).

Due to its static statistical nature, only one parameter can be adjusted to affect the resulting dataset: *energy level*. This defines how many Eigenvectors to use in the data transformation. A P x A matrix is created by aligning the P most significant Eigenvectors side-by-side. The algorithm for creating this matrix is shown below:

```
threshold = energy level   // ranges from 0 to 1
total energy = 0           // sum of eigenvalues
i = 0                      // eigenvector of interest
matrix = empty matrix      // resulting matrix

while total energy < threshold
    append eigenvector(i) as right column to matrix
    total energy = total energy + eigenvalue(i)
    i = i + 1
```

If the energy level is 1, all Eigenvectors will be used. If the energy level is 0, no Eigenvectors will be used, having no results. An energy level in between these two values will use P Eigenvectors.

The matrix that is created is then multiplied, using matrix multiplication, with the original A x B matrix, where B is the number of observations in the dataset. This will transform the data into a P x B matrix, with $P \leq A$. The number of dimensions in the dataset can be reduced significantly in this manner.

While this multiplication will make the values of each observation's features relative to one another, making them easier to group or classify, information is lost. The features of each observation have a different meaning than before. The original data can only be recovered if the P x B matrix created by the Eigenvectors is saved, along with the original mean of each feature (as described earlier).

4. Example Reduction of Dimensions

In this section, we show an example dataset to illustrate how ADRS and PCA can reduce features from any given dataset.

Consider a dataset of students in which three features are given: the student's height, extracurricular activities, and the amount of hours he/she studies per week. The classifier for each student would be his/her average GPA.

Preprocessing the data with ADRS will result in a classification table similar to Table 1. The table illustrates exactly how the features were merged. The hours of study, a continuous feature, was transformed into categorical data based on whether or not the student studied less than 5 hours. For the activities, the full time job would have one label, while the tutor and TA would share a common label. The height is not shown in the table below, thus being irrelevant. Each combination of possible relevant features is then classified with a particular outcome.

Hours of Study	Activities	Classifier
< 5 hours	Full Time Job	*GPA Range A*
< 5 hours	Tutor / TA	*GPA Range B*
>= 5 hours	Full Time Job	*GPA Range C*
>= 5 hours	Tutor / TA	*GPA Range D*

Table 1. Example ADRS Classification Table

If the same data was preprocessed with PCA, the outcome would be significantly different. The resulting dataset could contain 3, 2, or 1 dimensions, depending on the chosen energy level. Suppose an energy level is chosen such that the data becomes 2-dimensional. An example of such a transformation with PCA can be seen in Figure 2.

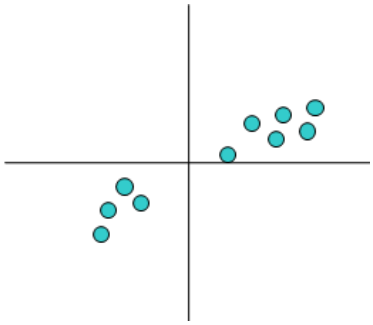

Figure 2. Example PCA Transformation

Each plot on the graph represents a different student. Each axis represents an Eigenvector used in the transformation of data. However, it is impossible to determine what features the dataset originally had. It is unclear how each of the three features of the dataset fit into this new mapping. Also, PCA does not classify each student. Instead, it suggests that each distinct group of students (in this case, two) can be expected to have a similar classifier (GPA).

5. Configuring the ANN

The ANN that was used in running the comparisons originated from the neural network library provided by MATLAB. The implementation was a feed-forward network using back-propagation that contains three layers: the input layer, one hidden later, and the output layer [3]. This made it easier for the network to analyze error results for each of the 4000 epochs (iterations), and modify the weights of the neurons accordingly.

The two variable parameters that were manipulated were the *learning rate* and *momentum constant*. Both of these parameters have significant effects on the back-propagation portion of the network, and have a value ranging from 0 to 1.

The learning rate determines how much the weights of each neuron change after each epoch. If the learning rate is too high, the network will become too sensitive to change in the error rates and will modify the weights of the neurons greatly. The network will become unstable. If the learning rate is too low, the network becomes significantly slow to converge.

The momentum constant allows the network to ignore small ripples in the change of error rates between epochs. A small momentum constant will make the network loop around a local error minimum. It will detect an increase in error between epochs, then modify the weights to decrease the error. The error will decrease once, but then increase again. This will cause the network to revert back to the old weights, thus repeating the process over again. A large constant will result in ignoring the error rates altogether, carrying incorrect results.

In order to appeal to all types of datasets and feed-forward neural networks, the four following configurations of learning rate (LR) and momentum constant (MC) were used:

1. LR = 0.5 ; MC = 0.1
2. LR = 0.5 ; MC = 0.3
3. LR = 0.9 ; MC = 0.4
4. LR = 0.9 ; MC = 0.6

6. Performance Comparison

A total of 15 datasets were used from the Machine Learning Repository, hosted by University of California, Irvine [4]. Each dataset was manipulated by both the ADRS and PCA prior to submission to the ANN. This created two groups of datasets. One group contained the results of each dataset preprocessed by ADRS, while the other contained the results produced by PCA. Figure 3 shows the experimental setup for comparing ADRS and PCA.

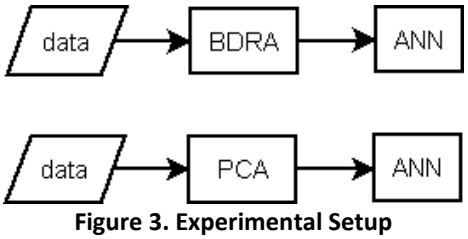

Figure 3. Experimental Setup

Both groups of 15 datasets were then processed by the ANN. A total of 180 iterations were completed on each dataset, on each of the 4 ANN configurations defined earlier. The tests were completed over a course of two weeks, using two dedicated servers. One server handled the data preprocessed by the ADRA, while the other server handled the data PCA transformed.

Each dataset that was sent through the network were partitioned into two categories: training data and test data. These two partitions were created at random in the beginning of each iteration. The training data accounted for 90% of the observations in the dataset, which was then used by the ANN to train on.

The remaining 10% of the data was considered to be the test data. This was the incoming data that the ANN would attempt to classify. The error rate for an iteration was calculated as the average percentage of test data incorrectly classified, either at the end of 4000 epochs through the ANN or after the network converged.

In order to keep the transformation between training and test data using PCA consistent, it was used to preprocess the training data prior to sending the data to the ANN. After PCA was applied, the same matrix formed by the Eigenvectors to manipulate the training data was used against the test data. The difference in the mean was also adjusted similar to the training data.

The results of the comparison tests can be seen in Table 2. For each dataset, the average error for the 180 iterations is calculated. A confidence interval of 95% is then calculated, and is shown in parenthesis.

UCI Data Set	ADRS + ANN	PCA + ANN
Annealing	8.87% (± 0.51%)	26.84% (± 0.76%)
Balance Scale	8.47% (± 0.31%)	20.02% (± 0.56%)
Credit	18.19% (± 0.64%)	39.81% (± 0.51%)
Ecoli	23.51% (± 0.63%)	28.42% (± 0.74%)
Diabetes	24.74% (± 0.37%)	30.93% (± 0.45%)
Glass	36.56% (± 0.75%)	58.54% (± 0.97%)
Hepatitis	21.37% (± 0.84%)	35.06% (± 1.08%)
Ionosphere	11.55% (± 0.46%)	32.98% (± 1.52%)
Iris	12.33% (± 0.89%)	13.02% (± 0.86%)
Lenses	46.04% (± 2.91%)	49.10% (± 2.44%)
Liver disorder	35.99% (± 0.63%)	38.83% (± 0.64%)
Lung cancer	43.47% (± 2.33%)	56.99% (± 2.18%)
Tic Tac Toe	20.74% (± 0.30%)	26.75% (± 0.63%)
Wine	6.29% (± 0.42%)	30.59% (± 0.79%)
Yeast	66.89% (± 0.82%)	62.04% (± 0.92%)

Table 2. Test Results

Across all datasets, the ANN produced an average error of 36.66% when the data was altered by the PCA. However, when ADRA was used as the preprocessor, the average error was reduced to 25.67%, a decrease of 10.99%.

The average length of time to run a single iteration on each dataset is illustrated in Table 3 below. The time measurement is in seconds, and a confidence interval of 95% is calculated and displayed in parenthesis.

UCI Data Set	ADRS + ANN	PCA + ANN
Annealing	168.83 (± 13.84)	133.38 (± 16.35)
Balance Scale	118.89 (± 5.80)	116.92 (± 5.71)
Credit	97.91 (± 12.97)	131.80 (± 7.47)
Ecoli	74.47 (± 5.58)	73.28 (± 4.05)
Diabetes	94.40 (± 27.08)	144.64 (± 10.71)
Glass	62.10 (± 4.22)	61.21 (± 3.81)
Hepatitis	43.05 (± 5.60)	53.25 (± 4.54)
Ionosphere	57.64 (± 5.35)	62.97 (± 4.98)
Iris	36.81 (± 6.99)	26.07 (± 2.40)
Lenses	51.31 (± 4.27)	24.34 (± 4.91)
Liver disorder	71.16 (± 4.62)	70.00 (± 4.14)
Lung cancer	52.56 (± 4.15)	4.22 (± 0.56)
Tic Tac Toe	237.20 (± 6.95)	202.81 (± 18.40)
Wine	56.57 (± 4.37)	57.87 (± 3.81)
Yeast	461.75 (± 37.86)	473.56 (± 24.17)

Table 3. Average Time to Run a Single Iteration

Out of the 15 datasets, ADRS performed with lower error rates than PCA on 12 of the datasets. For two of the datasets (iris and lenses), the averages of the errors were too close to be statistically significant. Only in the yeast dataset did PCA perform with lower error rates.

Considering the time to run each dataset through the ANN, ADRS performed significantly faster on only 2 of the datasets. PCA was able to be analyzed by the network significantly more quickly on 5 of the datasets as well. For the remaining datasets, the confidence intervals are too large to determine which preprocessor makes the ANN more efficient on time. Further research may reveal the causes behind this phenomenon.

7. Conclusion

In this paper, we have shown that ADRS performs with lower error rates than PCA when used as a preprocessor to ANN for data mining. This was shown by running benchmarks on the UCI repository datasets. Our results showed ADRS has better probability of error in 12 out of 15 datasets. The results also showed clearer understanding of the results because it contains the relevant features with their original names, which were retained by ADRS.

The results suggest that ADRS can become an alternative to PCA in preprocessing data. The computational load is slightly larger when using ADRS, but the lower error results and more intuitive output (via classification table) compensate for it. We suspect that ADRS performs with lower error rates because it is able to reduce the data while paying respect to nonlinear statistical dependencies amongst the features. PCA is limited in this respect in that it reduces the data based on no more than second order statistics. Thus, the ANN, after PCA preprocesses the data, receives less important information about the data.

One of the future directions is the comparison of performance of ADRS and ANN. In this paper, we have shown that ADRS has the potential for better performance by fine tuning the parameters used for running datasets. It was also shown in [5], where ADRS was compared with ANN. These parameters consist of number of continuous bins, missing value approach, search direction, and hold-out size. Exploration of the appropriate combinations of these parameters and optimized implementation may lead to better performance.

Another future direction is to implement ADRS in a distributed computing environment. This will allow ADRS to process very large volumes of data that are typical of applications in business, government, and medical fields. Currently ADRS runs on a stand-alone computer, which takes numerous days of computing to process 180 runs for each of the UCI datasets used in this paper. Running more datasets with very large number of runs will be prohibitively long and impractical. With the distributed system, ADRS can be tested on much more datasets more efficiently.

8. References

[1] R. S. Lynch and P.K. Willett, "Bayesian Classification and Feature Reduction Using Uniform Dirichlet Priors, " IEEE Tran. On Systems, Man, and Cybernetics, Vol. 33, No. 3, Jun 2003, pp. 448 – 464.

[2] Jolliffe, I.T., *Principle Component Analysis,* Springer, NY, 2002.

[3] C.C. Taylor D. Michie, D.J. Spiegelhalter. Machine Learning, Neural and Statistical Classification. Addison-Wesley Publishing Company, 1994.

[4] Asuncion, A. & Newman, D.J. (2007). UCI Machine Learning Repository [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science

[5] D. Patterson, F. Liu, D. Turner, A. Concepcion, and R. Lynch. Performance Comparision of the Data Reduction System. Proceedings of the SPIE Symposium on Defense and Security, Orlando, FL, March 2008.