A HADOOP-Based Framework for Parallel and Distributed Feature Selection

Victoria Hodge
The University of York
77 PUBLICATIONS **4,724** CITATIONS
SEE PROFILE

Tom Jackson
The University of York
44 PUBLICATIONS **462** CITATIONS
SEE PROFILE

Jim Austin
The University of York
263 PUBLICATIONS **6,300** CITATIONS
SEE PROFILE

A HADOOP-BASED FRAMEWORK FOR PARALLEL AND DISTRIBUTED FEATURE SELECTION

Victoria J. Hodge, Tom Jackson & Jim Austin

*Advanced Computer Architecture Group,
Department of Computer Science,
University of York, York, YO10 5GH, UK.*

{victoria.hodge,tom.jackson,jim.austin}@york.ac.uk

Abstract

In this paper, we introduce a theoretical basis for a Hadoop-based framework for parallel and distributed feature selection. It is underpinned by an associative memory (binary) neural network which is highly amenable to parallel and distributed processing and fits with the Hadoop paradigm. There are many feature selectors described in the literature which all have various strengths and weaknesses. We present the implementation details of four feature selection algorithms constructed using our artificial neural network framework embedded in Hadoop MapReduce. Hadoop allows parallel and distributed processing so each feature selector can be processed in parallel and multiple feature selectors can be processed together in parallel allowing multiple feature selectors to be compared. We identify commonalities among the four features selectors. All can be processed in the framework using a single representation and the overall processing can also be greatly reduced by only processing the common aspects of the feature selectors once and propagating these aspects across all four feature selectors as necessary. This allows the best feature selector and the actual features to select to be identified for large and high dimensional data sets through exploiting the efficiency and flexibility of embedding the binary associative-memory neural network in Hadoop.

Keywords

Hadoop; Distributed; Parallel; Data Fusion; Feature Selection; Binary Neural Network

1 Introduction

Frequently data contain too many features or too much noise [38] for accurate classification [10][22], prediction [10][15] or outlier detection [24][27] as only a subset of the features are related to the target concept (classification label or predicted value). Data from distributed systems may be intermittent and may contain duplicates as distributed systems communicate the data across a wide geographical area. Many machine learning algorithms are adversely affected by noise, omissions and superfluous features which can prevent accurate classification or prediction. Consequently, the data must be pre-processed by the classification or prediction algorithm itself or by a separate feature selection algorithm to prune these superfluous features [36][50]. For distributed systems this consists of pruning superfluous features either locally (at data source) or globally on an aggregated data set.

The benefits of feature selection include reducing the data size when superfluous features are discarded, improving the classification/prediction accuracy of the underlying algorithm where the algorithm is adversely affected by noise, producing a more compact and easily understood data representation and reducing the execution time of the underlying algorithm due to the smaller data size. For distributed systems, feature selection also translates into savings of power, hardware, and transmission as the data size is reduced. Feature selection can also minimise false positives by improving the data quality and thus the accuracy of the underlying classification or prediction algorithm.

In this paper, we focus on feature selection across all data for parallel and distributed classification systems. We aim to remove noise and reduce redundancy from the distributed network to improve classification accuracy. There is a wide variety of techniques proposed in the machine learning literature for feature selection including Correlation-based Feature Selection [17][18][20][21], Principal Component Analysis (PCA) [35], Information Gain [42], Gain Ratio [43], Mutual Information Selection [48], Chi-square Selection [39], Probabilistic Las Vegas Selection [40], Support Vector Machine Feature Elimination [16].

It is frequently not clear to the user which feature selector to use for their data and application. In their analysis of feature selection, Guyon and Elisseeff [15] recommend evaluating a variety of feature selectors. Hence, allowing the user to run a variety of feature selectors and then evaluate the feature sets chosen using their classification or prediction algorithm is highly desirable. Having multiple feature selectors available also provides the opportunity for ensemble feature selection where the results from a range of feature selectors are merged to generate the best set of features to use. Feature selection is a combinatorial problem so needs to run as efficiently as possible. We have previously developed a k-NN classification and prediction algorithm [26][28] using an associative memory (binary) neural network called the Advanced Uncertain Reasoning Architecture (AURA) [3]. This multi-faceted k-NN motivated a unified feature selection framework exploiting the speed and storage efficiency of the associative memory neural network. The framework lends itself to parallel and distributed processing across multiple nodes. This could be processing the data at the same geographical location using a single machine with multiple processing cores [47] or at the same geographical location using multiple compute nodes (parallel search) [47] or even processing the data at multiple geographical locations and assimilating the results (distributed processing) [5].

The main contributions of this paper are:

- To augment the AURA framework for parallel and distributed processing of data in Hadoop [1] [45],
- To describe four feature selectors in terms of the AURA framework. Two of the feature selectors have been implemented in AURA previously (but not using Hadoop) and two have not been implemented in AURA before,
- To analyse the resulting framework to show how the four feature selectors have common requirements
- To demonstrate distributed processing in the framework.

The feature selectors fit into one common data index representation. If we process any common elements only once and propagate these common elements to all feature selectors that require them then we can rapidly and efficiently determine the best feature selector and the best set of features to use for each data set under investigation. In section 2, we discuss AURA and related neural networks and how to store and retrieve data from AURA, section 3 demonstrates how to implement four feature selection algorithms in the AURA unified framework, section 4 describes parallel and distributed feature selection using AURA, we than analyse the unified framework in section 5 to identify common aspects of the four feature selectors and how they can be implemented in the unified framework in the most efficient way and finally, section 6 provides the overall conclusions from our implementations and analyses.

2 Binary Neural Networks

AURA [3] is an associative memory (binary) neural network. It is based on binary matrices, called Correlation Matrix Memories (CMMs) [6]. CMMs store associations between input and output vectors.

Input vectors address the CMM rows and output vectors address the CMM columns. Binary neural networks have a number of advantages compared to standard neural networks including rapid one-pass training, high levels of data compression, computational simplicity, network transparency, a partial match capability and a scalable architecture that can be easily mapped onto high performance computing platforms including parallel [47] and distributed platforms [5].

Previous parallel and distributed applications of AURA have included distributed text retrieval [23], distributed time-series signal searching [13] and condition monitoring [4]. We have previously developed a k-NN algorithm [26][28] using AURA called AURA k-NN. The AURA k-NN allows classification [31][32][37] and prediction [33] with feature selectors for both classification and prediction [30]. Thus, coupling feature selection, classification and prediction with the speed and storage efficiency of a binary neural network in the AURA framework allowing parallel and distributed data mining. This makes AURA ideal to use as the basis of an efficient distributed machine learning framework. A more formal definition of AURA, its components and methods now follows.

2.1 AURA

The AURA methods use binary input ***I*** and output ***O*** vectors to store records in a CMM ***M*** as in equation 1.

$$M = \bigvee_j I_j O_j^T \text{ where } \vee \text{ is logical } OR \qquad (1)$$

Training (construction of a CMM) is a single epoch process with one training step for each input-output association (each $\boldsymbol{I_j O_j^T}$ in equation 1) which equates to one step for each record in the data set. $\boldsymbol{I_j O_j^T}$ is an estimation of the weight matrix ***W(j)*** of the neural network as a linear associator with binary weights. Every synapse (matrix element) can update its weight independently and in parallel. This learning process is illustrated in Figure 1.

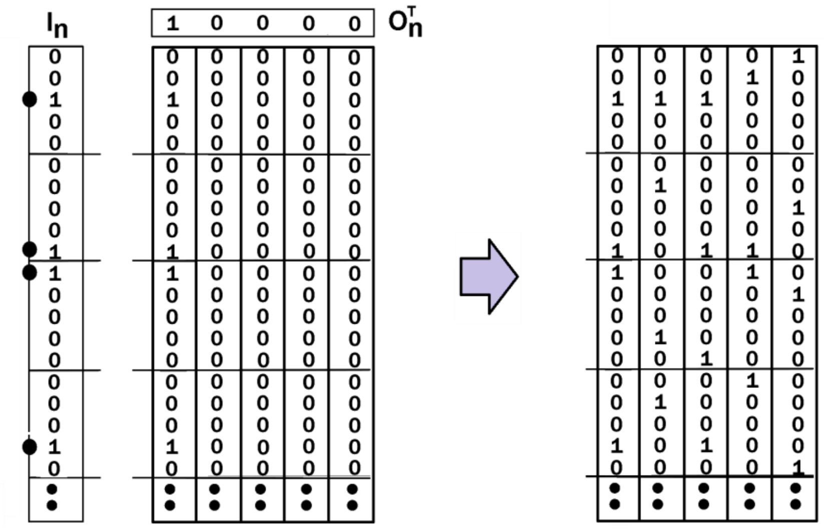

Figure 1 Showing a CMM learning input vector I_n associated with output vector O_n on the left. The CMM on the right shows the CMM after five associations $I_j O_j^T$. Each column of the CMM represents a record. Each row represents a feature value for symbolic features or quantisation of feature values for continuous features and each set of rows (shown by the horizontal lines) represents the set of values or set of quantisations for a particular feature.

For feature selection, the data are stored in the CMM which forms an index of all features in all records. The input vector and CMM rows represent the features and feature values; and the output vector and the CMM columns represent the data records. Note: for feature selection, the class values and the associated records that take those class values are also trained into the CMM as extra rows.

This process is identical to training the other feature values; the class is treated as an extra feature [30]. Figure 1 shows a trained CMM where each row is a feature or class value and each column represents a record. In this paper, we set only one bit in the vector $\boldsymbol{O_j}$ indicating the location of the record in the data set, the first record has the first bit set, the second record has the second bit set etc. Using a single set bit makes the length of $\boldsymbol{O_j}$ potentially large. However, exploiting a compact list representation [25] (more detail is provided in section 5) means we can compress the storage representation for single bit set vectors to a single index (set bit location), thus allowing AURA to be used for distributed processing with data sets of millions of records yet using a relatively small amount of memory.

2.2 Data

The AURA feature selector, classifier and predictor framework can handle symbolic, discrete numeric and continuous numeric features.

The raw data sets need pre-processing to allow them to be used in the binary AURA framework. Symbolic and numerical unordered features are enumerated and each separate token maps onto an integer (***Token*** $\mapsto$***Integer***) which identifies the bit to set within the vector. We refer to these features as symbolic henceforth. For example, a SEX_TYPE feature would map as (***F*** $\mapsto$ ***0***) and (***M*** $\mapsto$ ***1***). Any real-valued or ordered numeric features are quantised (mapped to discrete bins) [29], and each individual bin maps onto an integer which identifies the bit to set in the input vector. We refer to these features as continuous henceforth. Next, we describe the simple equi-width quantisation. We note that the Correlation-Based Feature Selector described in section 3.2 uses a different quantisation technique to determine the bin boundaries. However, once the boundaries are determined, the mapping to CMM rows is the same as described here.

To quantise continuous features, a range of input values for feature $\boldsymbol{F_j}$ map onto each bin. Each bin maps to a unique integer as in equation 2 to index the correct location for the feature in $\boldsymbol{I_j}$. In this paper, the range of feature values mapping to each bin is equal to subdivide the feature range into ***b*** equi-width bins across each feature.

$$\Re_{f_i} \rightarrow bins_{f_k} \mapsto Integer_{f_k} + offset(F_j) \qquad (2)$$
$$\text{where } f_i \in F_j \text{ and } cardinaltiy\left(Integer_{f_k}\right) \equiv cardinality\left(bins_{f_k}\right)$$

In equation 2, ***offset***($\boldsymbol{F_j}$) is a cumulative integer offset within the binary vector for each feature $\boldsymbol{F_j}$ and ***offset***($\boldsymbol{F_{j+1}}$) = ***offset***($\boldsymbol{F_j}$) + ***nBins***($\boldsymbol{F_j}$) where ***nBins***($\boldsymbol{F_j}$) is the number of bins for feature $\boldsymbol{F_j}$, $\rightarrow$ is a many-to-one mapping and $\mapsto$ is a one-to-one mapping.

```
For each record in the data set
  For each feature
    Calculate bin for feature value;
    Set bit in vector as in equation 2;
```

2.3 AURA Recall

To recall the matches for a query record, we firstly produce a recall input vector $\boldsymbol{R_k}$ by quantising the target values for each feature to identify the bins (CMM rows) to activate as in equation 3. During recall, the presentation of recall input vector $\boldsymbol{R_k}$ elicits the recall of output vector $\boldsymbol{O_k}$ as vector $\boldsymbol{R_k}$ contains all of the addressing information necessary to access and retrieve vector $\boldsymbol{O_k}$. Recall is effectively the dot product of the recall input vector $\boldsymbol{R_k}$ and CMM $\boldsymbol{M}$, as in equation 3 and Figure 2.

$$S^T = R_k^T \cdot M \qquad (3)$$

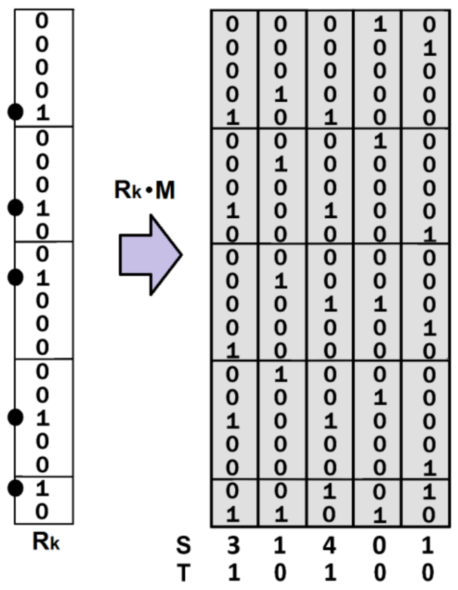

Figure 2 Showing a CMM recall. Applying the recall input vector R_k to the CMM M retrieves a summed integer vector S with the match score for each CMM column. S is then thresholded to retrieve the matches. The threshold here is either Willshaw with value 3 retrieving all columns that sum to 3 or more or L-Max with value 2 to retrieve the 2 highest scoring columns.

If $\boldsymbol{R_k}$ appeared in the training set, we get an integer-valued vector $\boldsymbol{S}$ (the summed output vector), composed of the required output vector multiplied by a weight based on the dot product of the input vector with itself. If the recall input $\boldsymbol{R_k}$ is not from the original training set, then the system will recall the output $\boldsymbol{O_k}$ associated with the closest stored input to $\boldsymbol{R_k}$, based on the dot product between the test and training inputs.

The AURA technique thresholds the summed output $\boldsymbol{S}$ to produce a binary output vector. For exact match, we use the Willshaw threshold [49]. This sets a bit in the thresholded output vector for every location in the summed output vector that has a value higher than or equal to a threshold value. The threshold varies according to the task. For partial matching, we use the L-Max threshold [9]. L-Max thresholding essentially retrieves *at least L* top matches. Our AURA software library automatically sets the threshold value to the highest integer value that will retrieve at least $\boldsymbol{L}$ matches.

Feature selection described in section 3 requires both exact matching using Willshaw thresholding and partial matching using L-Max thresholding.

3 Feature Selection

There are two fundamental approaches to feature selection [36][50]: (1) filters select the optimal set of features independently of the classifier/predictor algorithm while (2) wrappers select features which optimise classification/prediction using the algorithm. We examine the mapping of four filter approaches to the binary AURA architecture. Filter approaches are more flexible than wrapper approaches as they are not directly coupled to the algorithm and are thus applicable to a wide variety of classification and prediction algorithms. They can then be used as stand-alone feature selectors for other classification or prediction algorithms exploiting the high speed and efficiency of the AURA techniques to perform feature selection which is a combinatorial problem. We also intend to integrate them with the AURA k-NN for classification and prediction in the unified AURA framework.

We examine a mutual information based approach (**Mutual Information Feature Selection (MI)** detailed in section 3.1 that examines features on an individual basis, a correlation-based multivariate filter approach (**Correlation-based Feature Subset Selection (CFS)** detailed in section 3.2 that examines greedily selected subsets of features, a revised Information Gain approach **Gain Ratio (GR)** detailed in section 3.3 and a feature dependence approach **Chi-Square Feature selection(CS)** detailed in section 3.4 which is univariate. Univariate filter approaches such as Mutual Information or Chi-square are quicker than multivariate as they do not need to evaluate all combinations of subsets of features. The advantage of a multivariate filter compared to a univariate filter lies in the fact that a univariate approach does not account for interactions between features. Multivariate techniques evaluate the worth of feature subsets by considering both the individual predictive ability of each feature and the degree of redundancy between the features in the set.

During training for the MI, CFS, GR and CS algorithms, the input vectors $\boldsymbol{I_j}$ represent the feature and class values in the data records and are associated with a unique output vector $\boldsymbol{O_j}$. They all use an identical CMM and CMM training is an **n**-iteration process where *n* is the number of data records. This single CMM means that we can calculate and compare all four feature selectors using a single data representation. We note that the CFS as implemented by Hall [17] uses an entropy-based quantisation whereas we have used equi-width quantisation for the other feature selectors (MI, GR and CS). We plan to investigate unifying the quantisation as a next step. For the purpose of our analysis in section 5, we assume that all feature selectors are using identical quantisation.

We assume that all records are to be used during feature selection.

3.1 Mutual Information Feature Selection

Wettscherek [48] described a mutual information feature selection algorithm. The mutual information between two features is ``*the reduction in uncertainty concerning the possible values of one feature that is obtained when the value of the other feature is determined*' '[48]. We introduced an AURA version of the MI feature selector in [34] and just provide a brief overview here.

For our feature selection, AURA excites the row in the CMM corresponding to feature value $\boldsymbol{f_i}$ of feature $\boldsymbol{F_j}$. This row is a binary vector (***BV***) and is represented by $\boldsymbol{BVf_i}$. A count of bits set on the row gives $\boldsymbol{n}(\boldsymbol{BVf_i})$ from equation 4 and is achieved by thresholding the output vector $\boldsymbol{S_k}$ from equation 3 at Willshaw value 1. AURA also excites the row in the CMM corresponding to class value ***c*** where the binary vector is denoted ***BVc***. Again, counting the number of bits set on the row as above gives ***n(BVc)*** from equation 4.

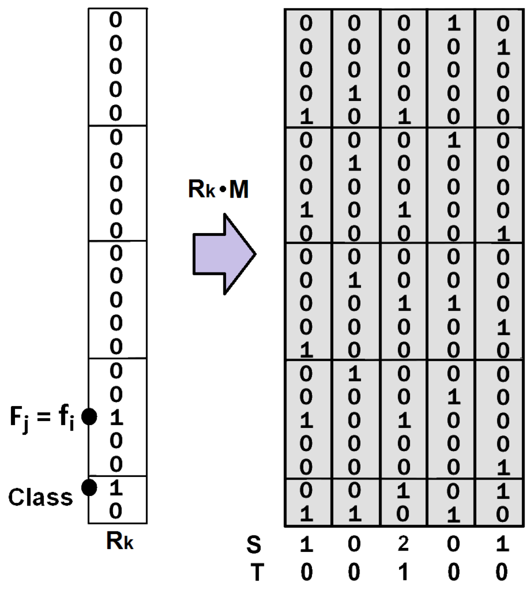

Figure 3 Diagram showing the feature value row and the class values row excited to determine co-occurrences.

If both the feature value row and the class values row are excited then the summed output vector will have a two in the column of every record with a co-occurrence of f_i with c_j as shown in Figure 3. By thresholding the summed output vector at a threshold of two, we can find all co-occurrences. We represent this number of bits set in the vector by ***n*****(*****BVf**$_i$ ∧ ***BVc***) which is a count of the set bits when ***BVc*** is logically ***AND***ed with ***BVf**$_i$* . The mutual information is given by equation 4 where ***rows(F**$_j$**)*** is the number of CMM rows for feature ***F**$_j$* and ***nClass*** is the number of classes:

$$I(C,F_j) = \sum_{i=1}^{rows(F_j)} \sum_{c=1}^{nClass} \frac{n(BVf_i \wedge BVc)}{N} \cdot log_2\left(\frac{\frac{n(BVf_i \wedge BVc)}{N}}{\frac{n(BVf_i)}{N} \cdot \frac{n(BVc)}{N}}\right) \quad (4)$$

We can follow the same process for real/discrete ordered numeric features in AURA. In this case, the mutual information is given by equation 5:

$$I(C,F_j) = \sum_{i=1}^{bins(F_j)} \sum_{c=1}^{nClass} \frac{n(BVb_i \wedge BVc)}{N} \cdot log_2\left(\frac{\frac{n(BVb_i \wedge BVc)}{N}}{\frac{n(BVb_i)}{N} \cdot \frac{n(BVc)}{N}}\right) \quad (5)$$

where ***bins*****(*****F**$_j$**)** is the number of bins (effectively the number of rows) in the CMM for feature ***F**$_j$* , ***N*** is the number of records in the data set, ***BVb**$_i$* is a binary vector (CMM row) for the quantisation bin mapped to by feature value ***f**$_i$*, ***BVc*** is a binary vector with one bit set for each record in class ***c***, ***n*****(*****BVb**$_i$* ∧ ***BVc***) is a count of the set bits when ***BVc*** is logically ***AND***ed with ***BVb**$_i$* (as shown in Figure 3 and ***n*****(*****BVc*****)** is the number of records in class ***c***.

The MI feature selector assumes independence of features and scores each feature separately so it is the user's prerogative to determine the number of features to select. The major drawback of the MI feature selector along with similar information theoretic approaches, for example Information Gain, is that they are biased toward features with the largest number of distinct values as this splits the training records into nearly pure classes. Thus, a feature with a distinct value for each record has a maximal information score. The CFS and GR feature selectors described next make adaptations of information theoretic approaches to prevent this biasing.

3.2 Correlation-based Feature Subset Selection

Hall [17] proposed the Correlation-based Feature Subset Selection (CFS) which measures the strength of the correlation between pairs of features. CFS favours feature subsets that contain features that are highly correlated to the class but uncorrelated to each other to minimise feature redundancy. CFS is thus based on information theory measured using Information Gain. However, as noted in the previous section, Information Gain is biased toward features with the largest number of distinct values. Hence, Hall and Smith [20] used a modified Information Gain measure (Symmetrical Uncertainty) to estimate the correlation between features given in equation 6. Symmetrical Uncertainty effectively normalises the value in the range [0,1] where two features are completely independent if ***SU=0*** and completely dependent if ***SU=1***.

$$SU(F_j, G_l) = 2.0 \cdot \left[\frac{Ent(F_j) - Ent(F_j \mid G_l)}{Ent(F_j) + Ent(G_l)}\right] \quad (6)$$

where the entropy of a feature ***F**$_j$* for all feature values ***f**$_i$* is given as equation 7:

$$Ent(F_j) = -\sum_{i=1}^{n(F_j)} p(f_i) log_2(p(f_i)) \quad (7)$$

and the entropy of feature ***F**$_j$* after observing values of feature ***G**$_l$* is given as equation 8:

$$Ent(F_j \mid G_l) = -\sum_{k=1}^{n(G_l)} p(g_k) \sum_{i=1}^{n(F_j)} p(f_i \mid g_k) log_2(p(f_i \mid g_k)) \tag{8}$$

Any continuous features are discretised using Fayyad and Irani's entropy quantisation [12]. The bin boundaries are determined using Information Gain and these quantisation bins map the data into the AURA CMM as previously.

As noted previously, for feature selection, the class values and the associated records that take those class values are also trained into the CMM as extra rows (extra features) as shown in Figure 3. CFS has many similarities to MI through calculating the values in equations 6, 7 and 8 and through using the CMM as noted below.

In the AURA CFS, for each pair of features (***F_j*,*G_l***) to be examined, the CMM is used to calculate ***Ent***(***F_j***), ***Ent***(***G_l***) and ***Ent***(***$F_j \cdot G_l$***) from equation 6. There are three parts to the calculation.

1. ***Ent***(***F_j***)requires the count of data records for the particular value ***f_i*** of feature ***F_j*** which is ***n(BVf_i)*** in equation 4 for symbolic and class features and ***n(BVb_i)*** in equation 5 for continuous features. Similarly, ***Ent***(***G_l***) counts the number of records where feature ***G_l*** has value ***g_k***.
2. ***Ent***(***$F_j \cdot G_l$***) requires the number of co-occurrences of a particular value ***f_i*** of feature ***F_j*** with a particular value ***g_k*** of feature ***G_l*** as in equations 4 and 5 and Figure 3 except that CFS calculates ***Ent***(***$F_j \cdot G_l$***) between a feature and the class ***n($BVf_i \wedge BVc$)*** and ***n($BVb_i \wedge BVc$)*** as well as between pairs of features ***n($BVf_i \wedge BVg_k$)*** and ***n($BVb_i \wedge BVb_k$)***.

CFS determines the feature subsets to evaluate using forward search. Forward search works by greedily adding features to a subset of selected features until some termination condition is met whereby adding new features to the subset does not increase the discriminatory power of the subset above a pre-specified threshold value. The major drawback of CFS is that it cannot handle strongly interacting features [19].

3.3 Gain Ratio Feature Selection

Gain Ratio (GR) [43] is a new feature selector for the AURA framework. GR is a modified Information Gain technique and is used in the popular machine learning decision tree classifier C4.5 [43]. Information Gain is given in equation 9 for feature ***F_j*** and the class ***C***. Hall and Smith [20] used a modified Information Gain measure in their CFS feature selector to prevent biasing toward features with the most values. GR is an alternative adaptation which considers the number of splits (number of values) of each feature when calculating the score for each feature using normalisation.

$$Gain(F_j, C) = Ent(F_j) - Ent(F_j \mid C) \tag{9}$$

where ***Ent***(***F_j***) is defined in equation 7 and ***Ent***(***F_j | C***) is defined by equation 8. Then Gain Ratio is defined as equation 10:

$$GainRatio(F_j, C) = \frac{Gain(F_j, C)}{IntrinsicValue(F_j)} \tag{10}$$

where ***IntrinsicValue*** is given by equation 11:

$$IntrinsicValue(F_j) = \sum_{p=1}^{V} \frac{S_p}{N} log_2\left(\frac{S_p}{N}\right) \tag{11}$$

and ***V*** is the number of feature values ***(n(F_j))*** for symbolic features and number of quantisation bins ***n(b_i)*** for continuous features and ***S_p*** is a subset of the records that have ***F_j=f_i*** for symbolic features or map to the quantisation bin ***bin***(***f_i***) for continuous features.

To implement GR using AURA, we train the CMM as described in section 2.1 using a suitable quantisation for continuous features. This could be Fayyad and Irani's quantisation used in CFS or could be equi-width binning as described in section 2.2. We can then calculate ***Ent***($\boldsymbol{F_j}$) and ***Ent***($\boldsymbol{F_j | C}$) as per the CFS feature selector described in section 3.2 to allow us to calculate ***Gain***($\boldsymbol{F_j, C}$). To calculate ***IntrinsicValue***($\boldsymbol{F_j}$) we need to calculate the number of records that have particular feature values. This is achieved by counting the number of set bits ***n(BVf***$_i$***)*** in the binary vector (CMM row) for $\boldsymbol{f_i}$ for symbolic features or ***n(BVb***$_i$***)*** in the binary vector for the quantisation bin $\boldsymbol{b_i}$ for continuous features. We can store counts for the various feature values and classes as we proceed so there is no need to calculate any count more than once. The main disadvantage of GR is that it tends to favour features with low Intrinsic Value rather than high gain by overcompensating toward a feature just because its intrinsic information is very low.

3.4 Chi-Square Algorithm

We now demonstrate how to implement a second new feature selector in the AURA framework. The Chi-Square (CS) [39] algorithm is a feature ranker like MI and GR rather than a feature selector; it scores the features but it is the user's prerogative to select which features to use. CS assesses the independence between a feature ($\boldsymbol{F_j}$) and a class ($\boldsymbol{C}$) and is sensitive to feature interactions with the class. Features are independent if CS is close to zero. Yang and Pedersen [51] and Forman [14] conducted evaluations of filter feature selectors and found that CS is among the most effective methods of feature selection for classification.

Chi-Square is defined as:

$$\chi^2(F_j, C) = \sum_{i=1}^{b(F_j)} \sum_{c=1}^{nClass} \frac{N * (wz - yx)^2}{(w+y) * (x+z) * (w+x) * (y+z)} \qquad (12)$$

where ***b(F***$_j$***)*** is the number of bins (CMM rows) representing feature $\boldsymbol{F_j}$, ***nClass*** is the number of classes, ***w*** is the number of times $\boldsymbol{f_i}$ and ***c*** co-occur, ***x*** is the number of times $\boldsymbol{f_i}$ occurs without ***c***, ***y*** is the number of times ***c*** occurs without $\boldsymbol{f_i}$, ***z*** is the number of times neither ***c*** nor $\boldsymbol{f_i}$ occur. Thus, CS is predicated on counting occurrences and co-occurrences and, hence, has many commonalities with MI, CFS and GR.

- Figure 3 shows how to produce a binary output vector (***BVf***$_i$ ∧ ***BVc***) for symbolic features or (***BVb***$_i$ ∧ ***BVc***) for continuous features listing the co-occurrences of a feature value and a class value. It is then simply a case of counting the number of set bits (1s) in the thresholded binary vector ***T*** in Figure 3 to count ***w***.
- To count ***x*** for symbolic features, we logically subtract (***BVf***$_i$ ∧ ***BVc***) from the binary vector (***BVf***$_i$) to produce a binary vector and count the set bits in the resulting vector. For continuous features, we subtract (***BVb***$_i$ ∧ ***BVc***) from (***BVb***$_i$) and count the set bits in the resulting binary vector.
- To count ***y*** for symbolic features, we can logically subtract (***BVf***$_i$ ∧ ***BVc***) from (***BVc***) and count the set bits and likewise for continuous features we can subtract (***BVb***$_i$ ∧ ***BVc***) from ***BVc*** and count the set bits.
- If we logically ***OR*** (***BVf***$_i$) with (***BVc***), we get a binary vector representing ***(F***$_j$***=f***$_i$***)*** ∨***(C=c)*** for symbolic features. For continuous features, we can logically ***OR*** (***BVb***$_i$) with (***BVc***) to produce ***(F***$_j$***=bin(f***$_i$***))*** ∨***(C=c)***. If we then logically invert this new binary vector, we retrieve a binary vector representing ***z*** and it is simply a case of counting the set bits to get the count for ***z***.

As with MI, CS is univariate and assesses features on an individual basis selecting the features with the highest scores, namely the features that interact most with the class. The main issue with CS is that it does not take into account inter-feature interactions.

4 Parallel and Distributed AURA

In distributed systems, data may be processed locally at each parallel compute node minimising communication overhead or globally across all distributed nodes providing an *"overall"* data view [46].

4.1 Parallel

In Weeks et al. [47], we demonstrated a parallel search implementation of AURA. AURA can be subdivided across multiple processor cores within a single machine or spread across multiple connected compute nodes. This parallel processing entails "striping" the data across several parallel CMM subsections. The CMM is effectively subdivided vertically across the output vector as shown in

Figure 4. In the data, the number of features *m* is usually much less than the number of records *N*, hence, $m << N$. Therefore, we subdivide the data along the number of records *N* (column stripes) as shown in

Figure 4.

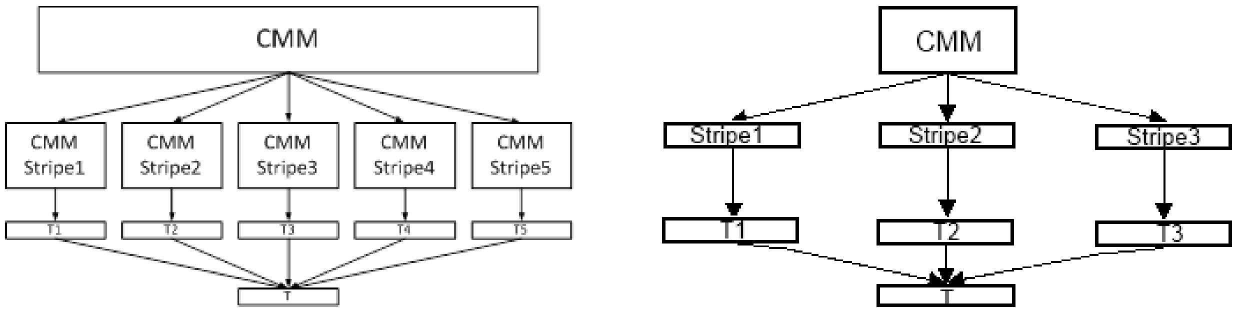

Figure 4 If a CMM contains large data it can be subdivided (striped) across a number of CMM stripes. In the left hand figure, the CMM is striped vertically (by time) and in the right hand figure the CMM is striped horizontally (be feature subsets). On the left, each CMM stripe produces a thresholded output vector T_n containing the top *k* matches (and their respective scores) for that stripe. All $\{T_n\}$ are aggregated to form a single output vector *T* which is thresholded to list the top matches overall. On the right, each stripe outputs a summed output vector S_n. All S_n are summed to produce an overall summed output vector which is thresholded to list the top matches overall.

Splitting the data across multiple CMM stripes using columns means that the CMM can store data as separate rows within a single stripe. Each record is contained within a single stripe. Each separate CMM stripe outputs a thresholded vector from that CMM stripe.

If the number of features is large then it is possible to subdivide the CMMs further. The CMM is divided vertically by the records (column stripes) as before and then the column stripes are subdivided by the input features (row stripes). Dividing the CMM using the features (row stripes) makes assimilating the results more complex than assimilating the results for column stripes. Each row stripe produces a summed output vector containing column subtotals for those features within the stripe. The column subtotals need to be assimilated from all row stripes that hold data for that column. Thus, we sum these column subtotals to produce a column stripe vector ***C*** holding the overall sum for each column in that stripe. Row striping involves assimilating integer vectors of length ***c*** where ***c*** is the number of columns for the column subdivision (column stripe).

4.2 Distributed

We have previously developed a distributed AURA search for condition monitoring of civil aerospace [5]. Here we use that as a basis for distributed feature selection. There are two central challenges for distributed feature selection: maintaining a distributed data archive so that data does not have to be moved to a central repository and secondly, orchestrating the search process across the distributed data.

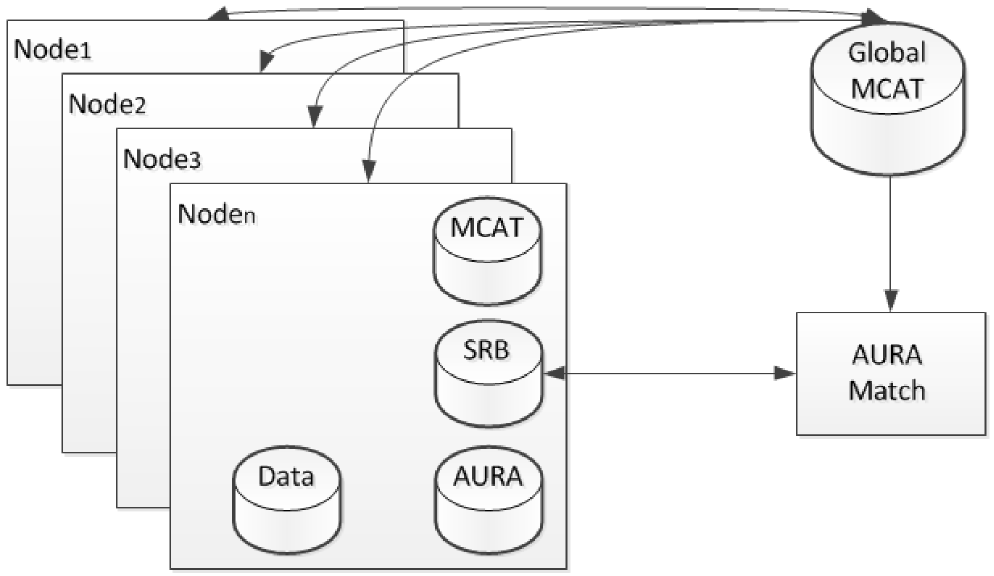

Figure 5 Distributed Data Management Architecture using a Pattern Match Controller accessing the Global Metadata Catalogue and Storage Resource Brokers on each processing node.

Our architecture for orchestrated search used a middleware stack (Pattern Match Controller [5] (PMC)) to farm search queries across distributed data resources. PMC allows a front-end service client to submit queries to all known data resources in a parallel, asynchronous manner, and to manage the processing and analysis of the data at the remote repositories. Forcing the pattern matching process to take place at the remote data repositories removes the costly requirement of moving large volumes of data during the search. An overview of the distributed AURA architecture is shown in Figure 5. The PMC uses a global Metadata Catalogue (global MCAT) that contains the locations of the data. In the implementation described in Austin et al. [5], the local data catalogue is provided by Storage Resource Broker [2] (SRB). SRB can manage distributed storage resources from large disk arrays to tape backup systems by mapping physical file locations to logical file handles referenced by PMC so PMC requires no knowledge of where the data resides. All nodes processing a query perform their search in parallel, searching all data held across the system as required. The PMC is also responsible for correlating the results and returning them.

Orchestrated search with minimal data movement can also be provided by the open source software project: Apache Hadoop [45]. Hadoop operates on the premise that "*moving computation is cheaper than moving data*" [1]. Hadoop allows the distributed processing of large data sets across clusters of commodity servers. It provides load balancing, is highly scalable and has a very high degree of fault tolerance. It is able to run on commodity hardware due to its ability to detect and handle failures at the application layer. When nodes fail with SRB, any search operations will not include results from these nodes. With Hadoop, there are multiple copies of the stored data so, if one server or node is unavailable, its data can be automatically replicated from a known good copy. If a compute node fails then Hadoop automatically re-balances the work load on the remaining nodes. There are two parts to Hadoop: MapReduce which assigns work to the nodes in a cluster and the Hadoop Distributed File System (HDFS) which is a distributed file system spanning all the nodes in the Hadoop cluster.

MapReduce divides (maps) the processing into separate chunks which are processed in parallel. The outputs of the processing tasks are combined (reduced) to generate a single result. The input and output data for MapReduce can be stored in HDFS on the same compute nodes used for processing the

MapReduce jobs. This produces a very high aggregate bandwidth across the cluster. The user's applications specify the input/output locations and supply *map* and *reduce* functions via implementations of appropriate interfaces and/or abstract-classes. The framework takes care of distributing the software/configuration, scheduling tasks, monitoring the tasks and re-executing any failed tasks.

HDFS links together the file systems on many local nodes to make them into one big file system. HDFS assumes nodes will fail, so it achieves reliability by replicating data across multiple nodes. Processing data in situ on local nodes is efficient compared to moving the data over the network to a single processing node. This local processing architecture of Hadoop has resulted in very good performance [44] on cheap computer clusters even with relatively slow network connections (such as 1 Gig Ethernet) [44]. Hence, Hadoop is ideal to underpin our distributed processing architecture.

5 Analysis

The key challenge for distributed feature selection is identifying the optimum method for distributing the search. Different data and applications will have different criteria that they wish to optimise. These could be optimising: communication overhead, processing speed, memory usage or combinations of these criteria. Hence, there is unlikely to be a single best technique for distribution.

However, Hadoop has demonstrated high performance for a wide variety of tasks [8]. It is aimed at batch processing tasks so is ideally suited to the task of feature selection where the feature selector is trained with the training data and feature selection is run once on a large batch of test data. In this paper, we focus on the implementation details of the four feature selectors using AURA with Hadoop. The capabilities of Hadoop have been demonstrated elsewhere [8] so we focus on describing how to map AURA CMMs to Hadoop to create an evaluation framework. Users will use this framework to select the best feature selector for their data and application using their own specific criteria.

Feature selection is a two part procedure. A training phase trains the data into the CMMs. A test phase then applies test data to the trained CMMs and correlates the results to produce feature selections. Each compute node holds a CMM that stores all local data. Data are trained into the CMM as described in section 2.1. During training, CMMs are not immutable as each association in equation 1 changes the underlying CMM so Hadoop MapReduce is not a suitable paradigm for CMM training. Hence, the CMMs are trained in a conventional fashion and uploaded to the Hadoop Distributed File System once trained. If the data stored in a node's CMM exceed the memory capacity of that node then the CMM is subdivided into stripes as described in section 4.1. The set of all CMM stripes at a node stores all data for that node. Every CMM stripe across the distributed system has to be coordinated so that record IDs (such as timestamps) are matched. If column 2 of one CMM stripe represents a time-stamp 10:00am on 31st May then column 2 in every associated row stripe must represent the identical time-stamp. When the results from different CMMs are unified then the columns from the various CMMs need to be aligned and hence must be identical time representations. The system is very flexible, we can access which nodes we require to access: all data or just a subset of data. The approach is a combination of the striping described above in section 4.1 and the CMM distribution described in section 4.2 with Hadoop orchestrating the search.

While the CMMs are being trained it is expedient to generate a MapReduce input file of input vectors to be used to produce the feature selections. These files will be split into batches by the MapReduce software and the results will be correlated to produce the feature selection scores. There is one input file per CMM stripe and the input vectors in each file represent the set of input vectors for recall to produce the feature selections.

Each CMM stripe that receives a search request, executes the recall process described in section 2.3. This retrieves a set of candidate matches as a vector. The candidate matches are the set of stored patterns that are close to the query in the feature space. In Hadoop the processing is coordinated by MapReduce [45]. Hadoop schedules the MapReduce tasks independently of the problem being solved. There is one Map job for each input file. Therefore, we model feature selection as a series of MapReduce jobs with each job representing one CMM stripe and the tasks are batches of file iterations (batch processing subsets of records) from the test data. The tasks are processed in parallel on distributed nodes. Each CMM stripe is read into a job. The recall function for CMM stripes is written as a Map task. Each MapReduce job invokes multiple Map tasks, each task represents a batch of recalls for a subset of input records, the batches execute in parallel. The Hadoop Mapper keeps track of the output vector versus record ID pairs so we know which output vector is associated with which record. The Reduce tasks perform the integer output vector thresholding as described in section 2.3 and write the data back into the file associated with CMM stripe. Multiple feature selectors can be run in parallel, each executing as a series of MapReduce jobs. The CMMs for feature selection are immutable so subsequent iterations do not depend on the results (or changes) of the CMMs.

This whole MapReduce process has to be coordinated. If the MapReduce process is running at a single location then it can be coordinated as a Java class that initiates the individual jobs and then coordinates the results from all jobs to produce the feature selection scores. If the processing is geographically distributed then it needs a more complete coordinator. This can be achieved using for example the UNIX curl command and a monitor process that determines when curl has collected new data. Alternatively, it can be achieved using a distributed stream processor such as Apache S4 (http://incubator.apache.org/s4/) or Twitter's STORM (https://github.com/nathanmarz/storm). Essentially, whichever tool is used this is a three part process: initiate the feature selection process at each of the distributed nodes; retrieve the results data from the distributed nodes; and, monitor when the results have been returned from all nodes and combine them into a single unified result. Each CMM stripe can return its results as

1. an integer vector $\boldsymbol{S_k}$ (unthrehsolded),
2. a thresholded vector $\boldsymbol{T_k}$ or
3. a list of the set bits in the thresholded vector.

Option 1 is the least efficient as, potentially, every column could have an integer score so the vector would be an integer vector of length *n* where *n* is the number of data records stored. This integer vector can be thresholded for option 2 which produces a binary vector. A binary vector requires less storage capacity than an integer vector (1 bit per element for the binary vector compared to 16 or 32 bits per element for the integer vector). For option 3, we would return a list of the set bits. For this we can exploit a compact list representation for representing binary vectors, more detail is given in [25]. This compact list representation is similar to the pointer representation used in associative memories [7]. It ensures that retrieval is proportional to the number of set bits in the thresholded output vector so is fast and scalable. For example, if we retrieved the binary thresholded vector 000000001011, with a compact representation this can be stored as {8,10,11} indexing from 0. On a vector of this size, the advantage of the compact representation is not apparent but if there were 100,000 records in the data set then the vector would have 100,000 elements. If only three records match (records; 8, 10 and 11) then storing {8,10,11} as indices requires much less memory and is much more efficient than storing 100,000 binary digits. Hence, wherever possible we use option 3. Minimising the communication overhead is of paramount importance in parallel and distributed processing systems so option 3 is best in this respect. All three methods require an index of what data are stored where and what each datum represents so the coordinating node can coordinate the matching, receive all matching data and determine the set of best matches across all searchable data.

The feature selection process produces a large set of output vectors from the CMM stripes; namely, all vectors necessary for all feature selectors. These results need to be amalgamated for each feature selector to produce the feature scores for that feature selector. Each feature selector will have a separate amalgamate program running at the coordinating node. This program uses the required vectors and set bit counts returned from AURA to produce the feature score as described in sections 3 and 5.1.

AURA has demonstrated superior training and recall speed compared to conventional indexing approaches [25] such as hashing or inverted file lists which may be used for data indexing. AURA trains 20 times faster than an inverted file list and 16 times faster than a hashing algorithm. It is up to 24 times faster than the inverted file list for recall and up to 14 times faster than the hashing algorithm. AURA k-NN has demonstrated superior speed compared to conventional k-NN [27] and does not suffer the limitations of other k-NN optimisations such as the KD-tree which only scales to low dimensionality data sets [41]. We showed in [34] that using AURA speeds up the MI feature selector by over 100 times compared to a standard implementation of MI. Feature selection is a combinatorial problem so a fast and efficient platform will allow rapid analysis of large and high dimensional data sets.

5.1 Distributed Feature Selection

All four feature selection algorithms have been well documented elsewhere and their relative strengths and weaknesses analysed on a range of data sets. Hence, we refer the reader elsewhere for accuracy evaluations. We note from the evaluations in the literature that no feature selector excels across all data sets. Different feature selectors are required for different data sets and applications. Hence, we are building a framework to allow the relevant feature selector to be chosen. Guyon and Elisseeff [15] posit that users should compare several feature selection methods before deciding the best for their problem. Therefore, we propose that users exploit the AURA framework to evaluate multiple feature selectors in parallel for use with either the integrated AURA k-NN classifier or predictor or for use with their own methods. Each feature selector produces feature scores. Some feature selectors such as CFS select the best set of features to use while other such as MI, CS and GR rank the features with the scores. For these feature rankers, the best set of features may then be chosen by the user, for example, using greedy search [50]. The user's classifier or predictor can then be evaluated against the sets of features to find the best feature selector and best subset of features according to the user's criteria. Having multiple feature selectors in a single framework also provides the opportunity for ensemble feature selection where the results from a range of feature selectors are merged to generate a consensus overview of the best set of features to use.

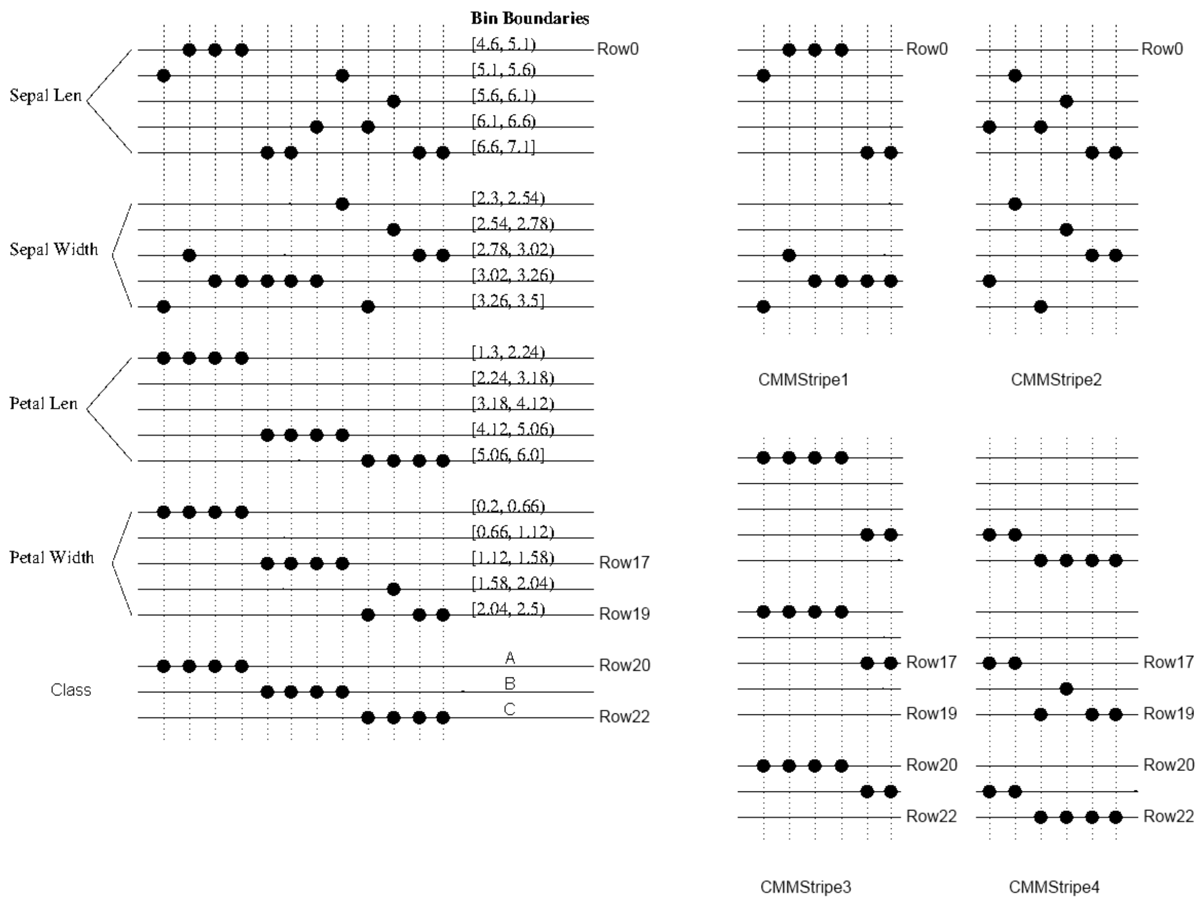

Figure 6 The 12 records from the iris data set, quantised and trained into a single AURA CMM (left) and subdivided across 4 stripes of the CMM (right). The letters in rows 20-22 indicate the class of the record: A=Iris-setosa, B=Iris-versicolor, C=Iris-virginica.

The feature selectors in section 3 have many commonalities when implemented in the unified AURA framework. We can demonstrate the commonalities by analysing 12 records from the Iris data set used in [34]. The data are illustrated in Figure 6 (left) when trained into the CMM. The 12 records have been trained into a CMM using the four features and the class. Each feature is continuous-valued and has been subdivided into five quantisation bins of equal width. Figure 6 (right) shows the same data divided into four CMM stripes (***CMMStripe1, CMMStripe2, CMMStripe3*** and ***CMMStripe4***). The horizontal (row-based) striping means that the features "*sepal len*" and "*sepal width*" are in the top stripes and "*petal len*", "*petal width*" and the *class* are in the bottom two stripes. The vertical (column-based) striping means that the first 6 data records are stored in the left two stripes and the other 6 records in the right two stripes. If the data were time-series or sequential, the column-based striping would form two time frames with the oldest data in the left two stripes and the newest data in the right two stripes. The input vectors are stored in a file for each CMM or CMM stripe. These files can then be batch processed in the Hadoop framework described. Within the evaluation, we consider how the data and CMMs would be accommodated in our Hadoop framework.

- MI, CFS, CS and GR can all use a single CMM representation for the data such as the CMM in Figure 6. This overall CMM is amenable to striping across the processing nodes to allow Hadoop processing in a similar fashion to
- Figure 4. This could encompass striping across local compute nodes or striping across geographically distributed nodes. Vertical striping would store different time frames at different nodes and horizontal striping would store different features at different nodes.

- MI, CFS, CS and GR all use ***BVf*** $_i$ (the binary vector where (F_j=f_i)), ***BVb*** $_i$ (the binary vector representing the quantisation bin ***bin(f*** $_i$***)***) and ***BVc*** (the binary vector representing all records that have a class label) so these only need to be extracted once and used in each feature selector as appropriate. For example in Figure 6, if we want all records where ***1.12 ≤ petal width < 1.58*** then we activate row 17 of the CMM using the input vector 0000000000000000100000 which will be one of the input vectors stored in the Hadoop input file for this CMM. We can then Willshaw threshold the integer output vector ***S*** at level 1 and retrieve the binary thresholded vector ***T*** with a bit set for every record where ***1.12 ≤ petal width < 1.58***, that is 000011110000 from Figure 6 (left). For the Hadoop distributed version, we use input files that hold the vectors to be processed. The vectors are subdivided across CMM stripes so that only the data that are relevant to that CMM stripe are input. Activate row 17 of ***CMMStripe3*** and ***CMMStripe4*** using the input vector 00000100000 as these stripes hold the relevant data. ***CMMStripe3*** will output 000011 and ***CMMStripe4*** will output 110000 in Figure 6 (right). These can be concatenated to form a single vector 000011110000. For a fully distributed system, each CMM stripe outputs a list of the indices of the set bits for the data relevant to that stripe.
- MI, CFS and GR all require a count of the number of data records where a particular feature has a particular value ***n(F*** $_j$***=f*** $_i$***)*** and a count of the number of records where the class has a particular label ***n(C=c)***. To count the number of records where ***1.12 ≤ petal width < 1.58***, we retrieve the binary thresholded vector as above (000011110000) and count the number of set bits giving four records. For the Hadoop approach, we coordinate the retrieval as above, concatenate the lists to produce a single overall list of set bits and count the list length.
- MI, CFS, CS and GR all use (***BVf*** $_i$ ∧ ***BVc***) and (***BVb*** $_i$ ∧ ***BVv***) for symbolic and continuous features respectively. For example, we can find all records where ***4.6 ≤ sepal len < 5.1*** and the class is ***A*** by activating rows ***0*** and ***20*** of the CMM using the input vector 1000000000000000000100 stored in the Hadoop input file for this CMM, thresholding at Willshaw level 2 to retrieve all records that match both inputs and retrieving the binary thresholded vector ***T***, 011100000000 from Figure 6 (left). This takes more coordinating in the Hadoop framework as the data for the feature value may not be stored with the data for the class; they may be in different CMM stripes. Activate row ***0*** in ***CMMStripe1*** and ***CMMStripe2*** using input vector 1000000000. Activate row ***20*** in ***CMMStripe3*** and ***CMMStripe4*** using input vector 0000000000100. The coordinating program needs to correlate the sections of the vector for the feature value and correlate the sections of the vector for the class to form a single vector. ***CMMStripe1*** needs to be added (summed) with the output integer vector of ***CMMStripe3*** (011100+111100=122200) and ***CMMStripe2*** needs to be added (summed) with the output integer vector of ***CMMStripe4*** (000000+000000=000000). The summed vector can then be thresholded at 2 giving 011100 for ***Stripe1+Stripe3*** and 000000 for ***Stripe2+Stripe4*** in Figure 6 (right). These two thresholded output vectors are concatenated to produce 011100000000. If the thresholded vectors are stored as lists of indices then this is simply a task of finding the common indices between the two vectors.
- MI, CFS, CS and GR all also need a count of the conjunction, that is ***n***(***BVf*** $_i$ ∧ ***BVc***) and ***n***(***BVb*** $_i$ ∧ ***BVc***) for symbolic and continuous features respectively. Hence, to count the number of records where ***2.04 ≤ petal width < 2.5*** and the class is ***C***, we retrieve the binary thresholded vector ***T*** (000000001011) and count the set bits giving 3 records. We coordinate the retrieval as above and, finally, count the number of set bits in the result.

Hence, rather than calculating these elements multiple times, we can take advantage of the commonalities by calculating each common value, binary vector or count only once and propagating the result to each feature selector that requires it. Following the common calculations, all necessary calculations will have been made for MI and GR. CFS just requires the pairwise feature versus feature analyses (***BVb*** $_i$ ∧ ***BVb*** $_k$). These are performed in the same way as the feature versus class analyses above. CS requires the manipulation of some of the binary vectors to produce the logical OR vectors.

This requires the coordination of the vectors. To find ($\boldsymbol{BVb_i}$) ∨ ($\boldsymbol{BV_c}$), we combine the list of set bits for ($\boldsymbol{BVb_i}$) with the list of set bits for ($\boldsymbol{BV_c}$) and count the resulting list length. By calculating the common elements first, the remainder of the calculations can be performed for each feature selector using either this CMM and processing the algorithms in series or by generating multiple copies of the CMM and processing them in parallel if sufficient processing capacity is available.

For the Iris data set, there are $\boldsymbol{n}(\boldsymbol{BV_{bi}})$ = ***20*** * $\boldsymbol{BVb_i}$ calculations (20 row activations) and $\boldsymbol{n}(\boldsymbol{BV_c})$ = ***3*** * ***BVc*** calculations. To calculate ($\boldsymbol{BVb_i}$ ∧ ***BVc***) requires $\boldsymbol{n}(\boldsymbol{BV_{bi}})$ x $\boldsymbol{n}(\boldsymbol{BV_c})$ = ***20*** x ***3*** =***60*** calculations. Hence, there are **83** common calculations (20+3+60) across all four feature selectors. CFS then needs to calculate ($\boldsymbol{BVb_i}$ ∧ $\boldsymbol{BVb_k}$) which would require 19! calculations if every feature value was compared to every other. However, CFS uses greedy forward search so that the number of comparisons is minimised [17] to a worst case of ($\boldsymbol{(20^2-20)/2=190}$). We have already extracted all ***20*** * $\boldsymbol{BVb_i}$ binary vectors so CFS needs 190 logical ANDs but no CMM accesses. We have saved a minimum of $\boldsymbol{n}(\boldsymbol{BV_{bi}})$ = ***20*** CMM accesses to extract the 20 binary feature vectors $\boldsymbol{BVb_i}$ and a maximum of 190 CMM accesses for worst case forward search. Manipulating the binary vectors can be performed at the coordinating node and in parallel as a Hadoop batch process. CS requires the logical OR vectors ($\boldsymbol{BVb_i}$ ∨ ***BVc***). Again, we already have all 20 $\boldsymbol{BVb_i}$ binary vectors and all 3 ***BVc*** binary vectors so there are 20 x 3=60 logical ORs to perform. Thus, we have saved a minimum of $\boldsymbol{n}(\boldsymbol{BV_{bi}})$ + $\boldsymbol{n}(\boldsymbol{BV_c})$ = ***23*** CMM accesses and potentially 60 CMM accesses if all 60 OR operations were performed in the CMM. Thus MI requires 83 calculations, GR also requires 83, CFS requires 83 plus 190 and CS requires 83 plus 60. In total there are ***83+83+83+190+83+60*** calculations. We have reduced this to ***83+190+60*** and the ***190+60*** additional calculations (not common) can use vectors already extracted so there is no need to access the CMM. We have saved ***3 * 83=249*** recalls from the CMM by finding common aspects, have removed a minimum of ***20+23*** further CMM recalls and have reduced the other calculations to logical operations on stored binary vectors. The minimum saving on CMM recalls is given by equation 13.

$$Recall\ Saving = \left(3\ \times \left(n(BVb_i) + n(BVc) + \left(n(BVb_i)\ \times n(BVc)\right)\right)\right) + \left(\left(2\ \times n(BVb_i)\right) + n(BVc)\right) \tag{13}$$

Once all of the binary vectors have been retrieved by the distributed Hadoop system, they need to be processed to calculate the feature scores using the various feature selectors. A coordinator program organises this and can be used to process in parallel. It takes the binary vectors stored in the files output from the Hadoop processing and uses these binary vectors to calculate the feature scores as per section 3. There is one feature score calculation process per feature selector (currently four feature selectors are described here). There is scope for subdividing the feature score calculations for each feature selector using parallel and distributed processing as appropriate to the location of the binary vectors required.

6 Conclusion

In this paper we have introduced a distributed processing framework for machine learning using the AURA neural network. There are currently four feature selectors available which may be used independently or coupled with the AURA k-NN for classification or prediction. All four feature selectors can use a single trained CMM. CMMs lend themselves to distributed processing as they can be striped (split) using both row-based and column-based striping. We have identified common aspects of the four feature selectors when they are implemented in the AURA framework and indicated how these common aspects may be processed as a common block. All remaining aspects of the feature selectors can then be implemented in parallel using duplicate copies of the trained CMM as compute resources allow. The CMM created for feature selection can be used directly for the AURA k-NN for classification or prediction and any unwanted features (those not selected by the feature selection) can simply be ignored (masked off). Alternatively, the CMM can be retrained with only the required data if processing speed and memory usage at recall time are the primary concern.

The AURA neural architecture has demonstrated superior training and recall speed compared to conventional indexing approaches such as hashing or inverted file lists [25] and an AURA-based implementation of the MI feature selector was over 100 times faster than a standard implementation [34]. This allows rapid processing of feature selectors on large and high dimensional data sets. The user can then evaluate the feature sets chosen by the feature selectors against their own data to determine the best feature selector and the best set of features.

The technique is flexible and easily extended to other feature selection algorithms. We intend to introduce a ReliefR and an Odds Ratio feature selector among others. By implementing a range of feature selectors in a single framework, we can investigate ensemble feature selection where the results from a range of feature selectors are merged to generate a consensus overview of the best set of features to use.

We plan to use the feature selection framework that we have developed in this paper in conjunction with the AURA k-NN for traffic state [31][32][37], for traffic state prediction [33], for journey time prediction [30] and for condition monitoring [4].

7 References

[1] The Apache Software Foundation. Apache Hadoop Project [Online] (accessed 10/05/12) http://hadoop.apache.org/common/docs/r0.20.2/hdfs design.html, 2012.

[2] SDSC Storage Request Broker [Online] (accessed 10/05/12) http://www.sdsc.edu/srb/index.php/, 2012.

[3] J. Austin. Distributed associative memories for high speed symbolic reasoning. In R. Sun & F. Alexandre, editors, IJCAI '95 Working Notes of Workshop on Connectionist-Symbolic Integration: From Unified to Hybrid Approaches, pp. 87-93, Montreal, Quebec, August 1995.

[4] J. Austin, G. Brewer, T. Jackson & V. Hodge. AURA-Alert: The use of binary associative memories for condition monitoring applications. In Procs 7th Int'l Conf. on Condition Monitoring and Machinery Failure Prevention Technologies, vol. 1, pp. 699-711, Stratford-upon-Avon, England, 2010.

[5] J. Austin, R. Davis, M. Fletcher, T. Jackson, M. Jessop, B. Liang & A. Pasley. Dame: Searching large data sets within a grid-enabled engineering application. Procs IEEE - Special Issue on Grid Computing, 93(3):496-509, 2005.

[6] R. Beale & T. Jackson. Neural Computing: An Introduction. Institute of Physics Publishing, Bristol, U.K. and Philadelphia, PA, 1990.

[7] H. Bentz, M. Hagstroem & G. Palm. Selection of relevant features and examples in machine learning. *Neural Networks*, 2(4):289 - 293, 1997.

[8] D. Borthakur et al. Apache Hadoop goes Realtime at Facebook. In Procs 2011 Int'l Conf. on Management of Data, SIGMOD '11, pp. 1071-1080, New York, NY, USA, 2011.

[9] D. Casasent & B. Telfer. High capacity pattern recognition associative processors. *Neural Networks*, 5(4):251-261, 1992.

[10] M. Dash & H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3):131-156, 1997.

[11] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters, *Comms of the ACM*, 51(1): 107-113, 2008.

[12] U. Fayyad & K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In Procs Int'l Joint Conf. on Artificial Intelligence, pp. 1022-1029, 1993.

[13] M. Fletcher, T. Jackson, M. Jessop, B. Liang & J. Austin. The signal data explorer: A high performance grid based signal search tool for use in distributed diagnostic applications. In CCGrid 2006 - 6th IEEE Int'l Symp. on Cluster Computing and the Grid, pp. 217-224, Singapore, 2006.

[14] G. Forman. An extensive empirical study of feature selection metrics for text classification*. J. Mach. Learn. Res.*, 3:1289-1305, March 2003.

[15] I. Guyon & A. Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.,* 3:1157-1182, March 2003.

[16] I. Guyon, J. Weston, S. Barnhill & V. Vapnik. Gene selection for cancer classification using support vector machines. *Mach. Learn*., 46(1):389-422, 2002.

[17] M. Hall. Correlation-based Feature Subset Selection for Machine Learning. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

[18] M. Hall. Correlation-based feature selection for discrete and numeric class machine learning. In Procs 17th Int'l Conf. on Machine Learning (ICML '00), pp. 359-366, 2000.

[19] M. Hall & G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Trans. on Knowl. & Data Eng*., 15(6):1437-1447, November 2003.

[20] M. Hall & L. Smith. Feature subset selection: a correlation based filter approach. In Int'l Conf. on Neural Information Processing and Intelligent Information Systems, pp. 855-858, 1997.

[21] M. Hall & L. Smith. Practical feature subset selection for machine learning. In Procs of the 21st Australian Computer Science Conf., pp. 181-191, 1998.

[22] J. Han & M. Kamber. Data mining: concepts and techniques. The Morgan Kaufmann Series in Data Management Systems. Elsevier, 2006.

[23] V. Hodge. Integrating Information Retrieval and Neural Networks. PhD thesis, Department Of Computer Science, University of York, Heslington, York, UK, YO10 5DD, September 2001.

[24] V. Hodge. Outlier and Anomaly Detection: A Survey of Outlier and Anomaly Detection Methods. LAP LAMBERT Academic Publishing, 2011.

[25] V. Hodge & J. Austin. An Evaluation of Standard Retrieval Algorithms and a Binary Neural Approach. *Neural Networks*, 14(3), 2001.

[26] V. Hodge & J. Austin. A High Performance k-NN Approach using Binary Neural Networks. *Neural Network*s, 17(3):441-458, 2004.

[27] V. Hodge & J. Austin. A Survey of Outlier Detection Methodologies. *Artificial Intelligence Review, 22*: 85-126, 2004.

[28] V. Hodge & J. Austin. A binary neural k-nearest neighbour technique. *Knowl. Inf. Syst.,* 8(3):276-292, 2005.

[29] V. Hodge & J. Austin. Discretisation of Data in a Binary Neural k-Nearest Neighbour Algorithm. *Technical Report YCS-2012-473,* Department of Computer Science, University of York, UK, 2012.

[30] V. Hodge, T. Jackson & J. Austin. A binary neural network framework for attribute selection and prediction. In 4th Int'l Conf. on Neural Computation Theory and Applications - NCTA 2012, Barcelona, Spain, October 5-7 2012.

[31] V. Hodge, R. Krishnan, J. Austin & J. Polak. A computationally efficient method for on-line identification of traffic control intervention measures. In 42nd Annual UTSG Conf., University of Plymouth, UK, January 5-7 2010.

[32] V. Hodge, R. Krishnan, J. Austin & J. Polak. A computationally efficient method for online identification of traffic incidents and network equipment failures. In Transport Science and Technology Congress: TRANSTEC-2010, Delhi, India, April 4-7 2010.

[33] V. Hodge, R. Krishnan, J. Austin & J. Polak. Short-term traffic prediction using a binary neural network. In 43rd Annual UTSG Conf., Open University, Milton Keynes, UK, January 5-7 2011.

[34] V. Hodge, S. O'Keefe & J. Austin. A binary neural decision table classifier. *NeuroComputing*, 69(16-18):1850-1859, 2006.

[35] I. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 2nd edition, 2002.

[36] R. Kohavi & G. John. Wrappers for feature subset selection. *Artif. Intell. J.*, Special Issue on Relevance, 97(1-2):273-324, 1997.

[37] R. Krishnan, V. Hodge, J. Austin, J. Polak & T.C. Lee. On identifying spatial traffic patterns using advanced pattern matching techniques. In Procs of Transportation Research Board (TRB) 89th Annual Meeting (DVD-ROM: Compendium of Papers),Washington, D.C., January 10-14 2010.

[38] H. Liu, H. Motoda, R. Setiono & Z. Zhao. Feature selection: An ever evolving frontier in data mining. *J. Mach. Learn. Res.*, pp. 4-13, 2010.
[39] H. Liu & R. Setiono. Chi2: Feature selection and discretization of numeric attributes. In Procs IEEE 7th Int'l Conf. on Tools with Artificial Intelligence, pp. 338-391, 1995.
[40] H. Liu & R. Setiono. A Probabilistic Approach to Feature Selection - A Filter Solution. In Procs of 13th Int'l Conf. on Machine Learning (ICML'96), pp. 319-327, 1996.
[41] A. McCallum, K. Nigam & L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In Procs 6th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining (KDD-2000), 2000.
[42] J. Quinlan. Induction of decision trees. *Mach. Learn*., 1:81-106, 1986.
[43] J. Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1992.
[44] N. Rutman. Map/reduce on lustre, (white paper). Technical report, Xyratex Technology Limited, Havant, Hampshire, UK, 2011.
[45] K. Shvachko, K. Hairong, S. Radia & R. Chansler. The Hadoop distributed file store system. In Procs IEEE 26th Symp. on Mass Storage Systems and Technologies (MSST), 28 June, 2010.
[46] P. Varshney. Multisensor data fusion. *Electronics Comm. Engineering Journal*, 9(6):245-253, 1997.
[47] M. Weeks, V. Hodge & J. Austin. A hardware accelerated novel IR system. In Procs 10th Euromicro Workshop (PDP-2002), Gran Canaria, Jan. 9-11. IEEE Computer Society, CA, 2002.
[48] D. Wettscherek. A study of distance-based machine learning algorithms. PhD thesis, Department of Computer Science, Oregon State University, Corvallis, USA, 1994.
[49] D. Willshaw, O. Buneman & H. Longuet-Higgins. Non-holographic associative memory. *Nature*, 222:960-962, 1969.
[50] I.H. Witten & E. Frank. *Data Mining: practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann, 2000.
[51] Y. Yang & J. Pedersen. A comparative study on feature selection in text categorization. In Procs 14th Int'l Conf. on Machine Learning (ICML'97), pp. 412-420, 1997.