Multi-Classifier Systems - Review and a Roadmap for Developers

Romesh Ranawana
romesh.ranawana@comlab.ox.ac.uk

Vasile Palade
vasile.palade@comlab.ox.ac.uk

University of Oxford Computing Laboratory, Oxford, United Kingdom

April 24, 2006

Abstract

Multi-Classifier Systems (MCSs) have fast been gaining popularity among researchers for their ability to fuse together multiple classification outputs for better accuracy and classification. At present, there is a lot of literature covering many of the issues and concerns that MCS designers encounters. However, we found out that there isn't a single paper published thus far which presents an overall picture of the basic principles behind the design of multi-classifier systems. Therefore, this paper presents a current overview of MCSs and provides a road-map for MCS designers. We identify all the key decisions that a designer would have to make over the design of a MCS and list out the most useful options available at each decision making step. We also present a case-study of the MCS theoretical issues considered, and present informal guidelines for the selection of different paradigms, based on the properties and distribution of the data. We also introduce a novel optimization of the standard majority voting combiner which uses a genetic algorithm.

1 Introduction

1.1 Classification and Classifiers

Classification is the process of assigning presented information into classes and categories of the same type. Most day-to-day tasks, from identifying the correct bus ride on, to selecting an item from a lunch menu, involves the proper understanding of perceived information. This often involves the assignment of perceived information into known classes, for better understanding. A classifier is a computer based agent which can perform such a classification.

There are many computational algorithms that can be utilized for classification purposes. The Wikipedia website[1] provides a list of the most commonly used classification algorithms and also provides a brief description of each. These classifiers can be broadly divided into two categories: rule-based classifiers and

[1] http://en.wikipedia.org/wiki/Category:Classification_algorithms.

computational intelligence based classifiers. Rule-based classifiers are generally constructed by the designer, where the designer defines rules for the interpretation of detected inputs. This is in contrast to computational intelligence based classifiers, where the designer only creates a basic framework for the interpretation of data. The learning or training algorithms within such systems are responsible for the generation of rules for the correct interpretation of data.

These computational intelligence based classification algorithms can further be categorized as supervised or unsupervised methods. Supervised techniques involve the participation of a teacher or expert, who is responsible for "teaching" or "instructing" the classifier. This 'teacher' presents the algorithm with a set of inputs and informs the algorithm of the category or class to which it should be assigned to. The teacher also instructs the algorithm on how far a generated output is from the expected answer. This helps the learning algorithm alter its parameters in order to improve its classification performance. After training, the classifier is expected to classify previously unseen, but similar, inputs to the correct classes. The backpropogation algorithm used on neural networks is a prime example of such a methodology. [7] lists out and categorizes the most popular supervised learning techniques available and also provides a series of statistical metrics for their evaluation and comparison.

In contrast, unsupervised classifiers are not presented with prior knowledge during training. The algorithm is expected to examine the input parameters and to assign them to different classes based on similarity measures and clustering factors. Self-organizing maps ([23]), which are used in conjunction with neural networks, are an example of such a paradigm.

A learning paradigm utilizes a set of training examples of the form $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ for the projection of a function $f(x)$. The x values are usually vectors of real or discrete values of the form $< x_{i1}, x_{i2}, \ldots, x_{in} >$. The y values are the expected outputs for the given x values, and are usually drawn from a discrete set of classes. Consequently, the task of the learning paradigm involves the approximation of a function $f(x)$ to produce a classifier.

The output of a classifier can take many forms. A formal classification for the types of outputs which can be produced is presented below (introduced in [51]):

- *Abstract form*: The classifier outputs a single unique class. That is, the classifier only outputs the class to which it thinks the input belongs to.
- *Rank level*: The classifier ranks each output class according to its belief on which class the input data belongs to. The label of the class it believes to be the most probable is sent as the first choice.
- *Measurement level*: The classifier assigns a value depicting the belief that the classifier has of the input value belonging to each class. Thus, the output of the classifier is an array which contains these belief values. These classifiers are also known as *probabilistic classifiers*.

As mentioned above, the purpose of a learning paradigm is to approximate an input to output mapping. However, system designers often encounter two limitations within the design process, which can often affect the proper approximation of a function. Firstly, the presence of noise (imperfections) within the data set can adversely affect the approximation performance. If all input data perceived by a training paradigm is of excellent quality, then there is a good

probability that the paradigm will come up with a good approximation of the function. However, real-life situations are not as simple and straightforward. Most data sets are not of good quality and contain a substantial quantity of noise. Such erroneous data can mislead the training paradigm which in turn can lead to wrong approximations. Secondly, most training paradigms have very clear-cut limitations on their operation. For example, the rule of thumb for the proper training of a neural network is that the paradigm should be presented with at least 10 times as much data as there are connections within the network. Less data can lead to the neural network reaching local minima in its training error and, consequently, returning bad approximations on the function. A small sized neural network with 10 inputs and one hidden layer, containing just few hidden nodes (e.g. 5–6), will have at least 50 connections, which in turn leads to a requirement of at least 500 training samples for proper training. Most of the complex data sets currently being used are often of much higher dimensions and consequently require large networks for proper approximations, but training a large network may not be an easy task.

Due to the limitations mentioned above, it has been experimentally observed that the construction of a perfect classifier for any given task is often impossible. Therefore, the best that system designers have to work with is using more classifiers and paradigms which provide near approximations of the functions expected. In most cases, individual classifiers are built using the same machine learning methodology. When different machine learning paradigms are used to approximate the same function, the approximations generated may easilly vary due to different interpretations and processing of the data and noise being made. This diversity among the performance of different individual classifiers and learning paradigms have led to the development of the Multi-Classifier Systems (MCS), which attempt to combine the approximations of different individual classifiers, sometimes built using different training paradigms, to obtain better results. Such systems are analogous to a company board of directors, where the board is usually constituted of people who have different levels of qualifications and expertise. For example, a board is usually constituted of an economist, an accountant, a management consultant and a marketing consultant. It is very rare that a board will have one person who is specialized in all these fields of expertise. Therefore, the board is compelled to make decisions in consensus with all the members of the board. A decision making process of this sort, where the final decision is generated by combining the opinions of all the members of the board is exactly how a MCS works. Of course, there can be many variations to this theme, where different members of the board could be given extra decision making capabilities based on the type of decision to be made. Methods for the incorporation of such variations within MCSs are discussed later in this paper

Intuitively, it makes sense that a combination of classifiers or experts provides better results than a singular decision maker. However, this is dependent on how independent and diverse the individual classifiers are. If all the classifiers provide similar and correlated results, the aggregated result will not provide any improvement to the recognition process. Accordingly, the diversity among the selected classifiers has been recognized as one of the key design features within a successful multi-classifier. The incorporation and evaluation of diversity among individual classifiers is reviewed in Section 3.

1.2 Overview and structure of the paper

This paper provides a review of the field of Multi-Classifier Systems and presents simple details and instructions for the construction of a MCS. There is currently a lot of literature on MCS available. However, we have noted a big void in the literature currently available, in that there isn't a single paper which analyzes all the important aspects of MCS design. Many of the considerations and solutions presented in this paper are available elsewhere, but finding them and summarizing the results can often prove to be difficult and time consuming. Therefore, this paper is an attempt at collecting all important and recent information related to MCSs in one location and provides a basic roadmap for MCS designers.

We identified that the most important considerations needed for the proper construction of a MCS are threefold: the proper construction of diverse classifiers, the selection of a representative topology for classifier integration and the selection of a suitable combinatorial function. Section 2 formally introduces MCS and justifies their use. Section 3 presents methods for the diversification of individual classifiers and also reviews different metrics available for the measurement of diversity between the created classifiers. Section 4 presents the topological options available to the designer for connecting the classifiers created. Section 5 presents common combinatorial functions and groups them into categories. Section 6 compares and examines data dependent combinatorial strategies. Section 7 presents a case-study of a multi-classifier system application and experimentally demonstrates an increase in accuracy when suitably diverse classifiers are combined in a proper manner. This section also compares results obtained through the use of different combinatorial strategies and provides guidelines for the selection of each strategy. We also analyze a combinatorial function where the combiner learning algorithm selects the classifiers from a pool of classifiers for incorporation. Such methodologies are called data dependent selection classifiers.

2 Multi-classifiers – a primer

Section 1 states that a learning paradigm is used to 'approximate' a function by using a set of training data. The reason for the learned functions only being approximated during training lies in the presence of noise within most training data sets. Correspondingly, when different paradigms are used for the approximation of the same function, slight variances in the function produced can occur. This can result in slight differences within the results obtained by individual classifiers. Intuitively, it makes sense that when these different classifiers are combined together in a proper fashion, the combined result should be at least equal to the results of the best performing classifier within the ensemble of classifiers. Such a combination of diverse classifiers for the combined classification of data is known as a *multi-classifier*. Consequently, multiple-classifiers are being used to achieve the best possible classification, by increasing the efficiency and accuracy of classification. However, multi-classifiers will only work when it is possible to build individual classifiers which are more than 50% accurate (i.e. not random classifiers), and very important, are independent of each other([18]).

[18] claims that such an ensemble would only be more successful than an

individual classifier if the individual classifiers disagree with each other. Such independence among classifiers is known as the *diversity* within the ensemble. Detailed explanation on diversity and metrics used to measure it are discussed in detail within Section 3.2.

If all the classifiers within an ensemble are identical, then no improvement in the system can be obtained through combination. However, if the results obtained are different, proper combination can result in an improvement of the overall performance. This idea is graphically illustrated in Fig. 1. Within this figure, the shaded region represents the true positives within an output space, and the clear ellipses represent the true positive spaces identified by three separate classifiers. As depicted, typical classifier behavior would involve the proper classification of a large percentage of both true positives and true negatives and the improper classification of a small percentage. Conventionally, it is very ambitious to expect a designed classifier to properly classify the entire output space of a given input set. Therefore, it makes sense that the proper combination of the entire set or a subset of classifiers should provide the designer with an output space classification which is closer to the expected output space.

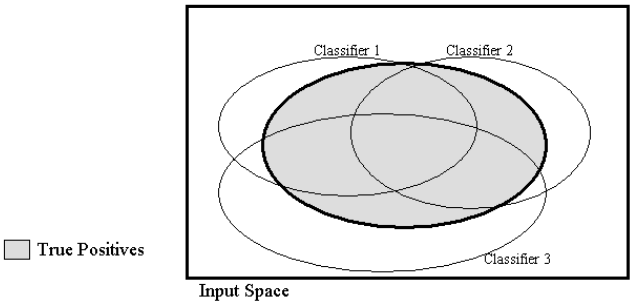

Figure 1: A classification problem with different classifiers being used for the same data set. The shaded region represents the true positives and the ellipses represent the space classified by each classifier as being true positives.

Individual classifier models are therefore being challenged by combined pattern recognition systems which often show better performance. Moreover, when a system which incorporates more methodologies (e.g. a hybrid system) is used, the inadequacies of one methodology can be nullified by the characteristics of another. That is, a multiple classifier can nullify the predictive inaccuracy obtained through the use of singular classification systems. The challenge in building such a system lies in the proper selection of a combination method and the proper diversification of the individual classifiers.

It can be proven that the performance of the entire ensemble of classifiers can be never worse than that of the best expert. [36, 40] provide results tested on protein secondary structure prediction. [1] also lists out an overview of applications in molecular biology. [6] indicates that the use of an ensemble of classifiers could achieve better recognition rates that those obtained through a singular classifier when:

1. The recognition rate of each individual classifier of the ensemble is greater than 0.5, and

2. Errors made by each individual classifier are uncorrelated.

[6] cites [18] and justifies this statement by considering three classifiers: $\{C_1, C_2, C_3\}$. Let x be a new input example. If the three classifiers are absolutely identical, then when $C_1(x)$ provides a wrong classification, $C_2(x)$ and $C_3(x)$ will also provide erroneous results. If the errors made by the three classifiers are uncorrelated, the case might be that when $C_1(x)$ is wrong, $C_2(x)$ and $C_3(x)$ may be correct. In such a case, the majority vote (See Section 5.1) among the three classifiers will correctly classify x. [6] goes on to state that if the error rates of R classifiers $C_j, j = 1, \ldots, R$, "are smaller than 0.5 and are independent, then the probability that the majority vote will be erroneous will be the area under the binomial distribution where more than $R/2$ hypothesis are wrong". Obviously, if the error rates exceed 0.5, even if the individual classifiers make uncorrelated errors, the error rate of the ensemble will increase with the combination of the classifiers.

[6] also states three causes as to why multiple classifiers exhibit better performance when compared to singular systems:

1. As mentioned in Section 1, a large search space needs a large amount of data for proper approximation. It is often the case that such amounts of data is unavailable, which in turn leads to bad approximations.

2. Most of the training algorithms used are heuristics. Heuristic algorithms always search for approximate solutions.

3. Most search algorithms do not have the capability to approximate the complex search spaces that a vast amount of the commonly used data sets occupy.

[6] further states that the use of multiple classifier solutions and their combinations lead to the nullification of most of these problems.

In a situation where a number of classifiers are available, the simplest approach would be to select the best performing classifier and use it for the classification task ([33]). This approach, although simple and easy to implement, does not guarantee good performance ([38]). It is highly probable that a combination of classifiers would outperform a singular classifier ([42]). On the other hand, different and worse performing classifiers might only add to the complexity of the problem and could even provide lower results than the worst classifier. Thus, it is a well known fact that if a multi-classifier system is to be successful, the different classifiers should have good individual performances and be sufficiently different from each other ([45]). Nevertheless, neither individual performances ([37, 54]) nor the diversity ([41, 47]) provides an ideal measure of how successful the combination of the classifiers will be.

As explained in [42], the core element of classifier selection is the selection criteria. The most natural choice is the combined performance, which could also be used as the criterion for the selection of the combiner. The only drawback of this methodology is the exponential complexity of testing out all possible combinations of a given set of classifiers. Evolutionary optimization techniques, like genetic algorithms and particle-swarm optimization, could be used to circumnavigate such problems [34].

There are three types of combiner functions. These three situations are a direct consequence of the three types of classifier outputs described in Section 1. They are:

1. Situations where the classifiers produce abstract outputs. Here, given R classifiers $C_j, j = 1, \ldots, R$, where each classifier assigns a label θ_j to a given input x. Therefore, the task of the multi-classifier involves the assignment of a definitive label (θ_{MCS}) to x.

2. When ranking classifiers are considered, for each input x, each classifier produces an integer array $RANK_j, j = 1, \ldots, R$. Each element within this array corresponds to one of the output classes. The array is usually sorted such that the most highest ranked output class is first on the array and the least ranked output class is at the end of the array. Therefore, the task of the multi-classifier is to produce an integer array ($RANK_{MCS}$) which ranks the output classes according to the given input x.

3. When classifiers which operate at the measurement level are considered, each classifier C_j produces a real vector of the form $M_j = [m_{1j}, \ldots, m_{cj}]$. Here, m_{ij} denotes the belief that the classifier j has that a certain input x belongs to class i, where $i = 1, \ldots, c$, and c is the number of output classes. Therefore, the task of the multi-classifier system is to build another real vector (Y_{MCS}) to denote its belief of the input x belonging to each output classes.

[29] states three classifier properties which are desirable for their incorporation within a multi-classifier. They are orthogonality, complementarity and independence. Orthogonality is not directly measurable, but is a rough measurement of how different the decisions are among the different classifiers. The complementarity measures the differences between the strengths and weaknesses of the different classifiers. Independence among classifiers measures the influence that one classifier has over another, over the decision making process, or how interrelated the results between two classifiers are.

A multi-classifier can be constructed either in a parallel, cascading or combined manner (see Section 4). The selection of a proper topology depends on the type of problem at hand. Some tasks can be divided into sub-tasks, and consequently, each sub-task can be implemented using a different classifier. Some tasks can be divided into different specialities depending on the number and diversity of output classes present. In such a case, specialized classifiers can be constructed to implement different specialities. However, the most common use of multi-classifiers involves the development of diverse classifiers which solve the same task and the combination of them in a parallel manner. Further details of these topologies are given in Section 4.

To summarize, the success of a multi-classifier system depends to a large extent on three key features: proper selection of classifiers, topology, and combinatorial methodology. The proper selection of classifiers involves the creation and selection of classifiers which are adequately diverse. The following three chapters provides a review of these design features and also presents the options which are available at each design step.

3 Diversity

As mentioned in Section 1, the successfulness of a MCS system depends to a large extent on the proper selection of diverse classifiers for incorporation. Therefore, the proper construction of classifiers which are suitably diverse from each other is of paramount importance. This section presents a collection of the most commonly used and popular diversification methods available, and also lists out a few metrics available in the literature for the measurement of the diversity among a pool of available classifiers.

3.1 Incorporating diversity among multiple classifiers

A classifier needs to be trained. This training process can be performed in either a supervised or unsupervised manner. Supervised learning involves the creation/approximation of a function from training data, by using a "teacher". The training data is composed of two main elements – input objects (typically vectors), and desired outputs. The output of the approximated function can either be a continuous value or a class label. The task of the teaching algorithm is to map the presented inputs onto the output classes. However, during training, the training algorithm is, more often than not, presented with only 'samples' from the input space, and is therefore not shown the entire data set. Consequently, the learner has to be able to generalize previously unseen data by using the presented data. These properties allow a MCS designer many opportunities for incorporating diversity within a collection of classifiers. The options include the diversification of inputs via feature selection, the use of multiple encoding methods, the randomization and optimization of the algorithm characteristics, and proper selection of input data from the entire training set.

Four diversification schemes presented below were first introduced formally in [6].

3.1.1 Subsampling the training set

This method requires the execution of the training algorithm several times on different subsets of the training set in order to obtain different hypotheses. Such method is useful for unstable training algorithms, that is, paradigms which induce large changes within the system's parameters in response to small changes within the data. Examples include Neural Networks and Decision Trees.

The most popular method for the selection of training set samples is known as the '*bagging*' technique. Given a training set of m samples, on each run of the training algorithm, an m amount of training samples are drawn randomly with replacement from the training set and presented to the training algorithm. It is clear that some training examples may appear more times in a drawn training set.

An alternative subsampling method involves the construction of the training set by leaving out disjoint subsets of the training data, in a similar manner as the traditional cross-validation method works. This can be achieved by, for example, first constructing q disjoint subsets, and then selecting data from each of the subsets to construct q overlapping subsets, by dropping out a different subset each run.

There is another very popular method that involves the manipulation of the training set to develop multiple hypotheses. This method would be similar to the bagging method if not for the condition that the sampler, at each iteration i, draws a training set of size m by sampling, with replacement, according to the probability distribution $p_i(x)$ over the training set. This sample set is then used to construct the classifier C_i. The error rate ϵ_i of C_i is then used to adjust the probability distribution over the training examples. This methodology is used successfully in the implementation of the ADABOOST algorithm ([11, 12]).

3.1.2 Manipulation of the input features

These techniques involves the manipulation of the set of input features to train different classifiers. Here, subsets of the total number of input features are selected in order to train multiple classifiers. This training methodology only works if the training set features are highly redundant. Such a methodology was successfully used in the implementation of the system described in [3], where the authors present a system with 32 trained neural networks trained on 8 data sets consisting of different feature subsets selected out of a 119 feature input space. In addition to this, the networks were further differentiated by the use of different network sizes.

[34] presents an alternative method for the diversification of classifiers via the inputs. They present the system MultiNNProm, where diversification is achieved by presenting the inputs to multiple neural networks via different encoding methods. Then, by using genetic algorithms, the networks are then adapted to suite each encoding method. The most suitable configuration is coupled with its representative encoding method and classifiers are constructed. The incorporation of these constructed classifiers within a MCS yielded over a 5% increase on the performance of the best individual classifier.

3.1.3 Manipulation of the output classes

This technique involves the manipulation of the output (y) values that are presented to the learning algorithm. This technique is particularly useful if the number of output classes is large. This allows the designer to divide the output classes into a smaller number of subsets of classes which could then be utilized for the training of a classifier. All classes in a subset are re-labeled with the same label before training. Thus, the trained classifier would be able to differentiate among the small number of classes it was trained on. By repeating this process more times, we obtain more classifiers.

3.1.4 Incorporation of randomness

This technique involves the injection of randomness into the training algorithm. An example of this method would involve the training of architecturally identical neural networks with different initial weights on the same data set. An example of such an implementation is presented in [34], where they incorporate a similar methodology by training diverse neural networks on the same data set.

3.1.5 Diversification of algorithm parameters

Another obvious and simple method for achieving diversification is accomplished by the incorporation of diversity within the training algorithm. All learning paradigms have some parameters which represent their behavior during training. For example, the backpropogation algorithm for training neural networks is represented by a number of hidden layers, the number of neurons per layer, a minimum error and a learning rate. These values could be varied on the same data set to achieve diverse results.

3.2 Measuring diversity among classifiers

Research on the measurement of diversity among classifiers within a collection of classifiers has intensified within the last few years. The Journal of Information Fusion recently published a special issue entitled 'Diversity in multiple classifier systems'([30]). Most literature concerning diversity is covered within this special issue. Therefore, this section only lists out the main diversity measurement schemes currently being used.

Combining similar classifiers does not make too much sense as the results will not be improved. Thus, the measurement of the diversity among a given set of classifiers is very important for the correct selection of classifiers to be incorporated within an ensemble. Measuring the diversity between two classifiers only is relatively not hard to be done ([27]). The measurements available become less conclusive when the number of classifiers exceeds three or more. At present, there is no strict computational definition for diversity. However, [27] refers and adapts a definition presented by biologists ([35], see Appendix) for the purpose of diversity measurement among classifiers.

[25] presents a list of 10 classifiers diversity measures, four of which are pairwise diversity measures and the rest non-pairwise. The following section lists out these 10 diversity metrics as the list comprehensively covers the main diversity measurement schemes currently being used. Further details on each of them could be found in [25].

3.2.1 Pairwise diversity measures

A pairwise diversity measure is used to measure the diversity between two given classifiers. Widely known statistical literature was used for the derivation of these diversity measures.

The Q statistic ([25, 53])

Let N denote the number of patterns in the training data set (denoted by X) and R be the number of classifiers. The output of classifier C_j, $j = 1, \ldots, R$, is represented by an N dimensional vector $M_j(X) = \{m_{1j}, \ldots, m_{Nj}\}$, where $m_{ij} = 1$ if C_j correctly recognizes the input pattern x_i, and 0 otherwise.[2] Because the notations used in [25] are very suggestive, and for easing the task of understanding for a prospective reader when following the references, we prefered to maintain some of the notations in the following subsections. Then, the Q statistic for two classifiers C_i and C_j is defined by Eq. (1), where N^{ab} (a and b 0 or 1) is the number of patterns

[2] Here, C represents the collection of classifiers and there exists c output classes.

x_k of X for which $m_{ki} = a$ and $m_{kj} = b$, $k = 1, \ldots, N$. If the two classifiers are independent, Q will be 0. The average Q statistic calculated over all pairs of classifiers in the collection of classifiers C is then defined by Eq. (2).

The correlation coefficient $-\ \rho$ ([25])

This measure presented in [25] can also be used for the evaluation of binary output classifiers and is given by Eq. (3).

The disagreement measure ([25, 44])

This measure was first introduced in [44] and, in the same notation as above, is the ratio between the number of cases in which one classifier is correct and the other is incorrect ($N^{01} + N^{10}$) to the total number of cases (in fact N), see Eq. (4).

The double fault measure ([15, 25])

[15] proposed this method to generate a pairwise diversity matrix for a collection of classifiers. This matrix is then used to select the classifiers within the collection which are least related. The double fault measure (DF) for two classifiers is defined as the ratio between the input examples which have been misclassified by both classifiers to the total number of examples in the training set (Eq. (5)).

Table 1: Pairwise diversity measures [25]

Name	Formula
Q Statistic	$$Q_{ij} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}} \quad (1)$$ $$Q_{avg} = \frac{2}{R(R-1)} \sum_{i=1}^{R-1} \sum_{j=i+1}^{R} Q_{i,j} \quad (2)$$
ρ	$$\rho_{ij} = \frac{N^{11}N^{00} - N^{01}N^{10}}{\sqrt{(N^{11} + N^{10})(N^{01} + N^{00})(N^{11} + N^{01})(N^{10} + N^{00})}} \quad (3)$$
Disagreement Measure	$$Dis_{ij} = \frac{N^{01} + N^{10}}{N^{11} + N^{10} + N^{01} + N^{00}} \quad (4)$$
Double Fault Measure	$$DF_{ij} = \frac{N^{00}}{N^{11} + N^{10} + N^{01} + N^{00}} \quad (5)$$

3.2.2 Non-pairwise diversity measures

Non-pairwise diversity measures are used to evaluate the diversity among three or more classifiers. [25] presents six such methods:

Table 2: Non-pairwise diversity measures [25]

Name	Formula
E	$E = \frac{1}{N}\sum_{j=1}^{N}\frac{1}{(R-\lceil R/2\rceil)}min\{l_j, R-l_j\}$ (6)
KW	$KW = \frac{1}{NR^2}\sum_{j=1}^{N} l_j(R-l_j)$ (7) $KW = \frac{R-1}{2R}Dis_{avg}$ (8)
κ	$\kappa = 1 - \frac{\frac{1}{R}\sum_{j=1}^{N} l_j(R-l_j)}{N(R-1)\overline{p}(1-\overline{p})}$ (9) $\kappa = 1 - \frac{R}{(R-1)\overline{p}(1-\overline{p})}KW = 1 - \frac{1}{2\overline{p}(1-\overline{p})}Dis_{avg}$ (10)
Generalized Diversity	$p(1) = \sum_{i=1}^{R}\frac{i}{R}p_i$ (11) $p(2) = \sum_{i=1}^{R}\frac{i(i-1)}{R(R-1)}p_i$ (12) $GD = 1 - \frac{p(2)}{p(1)}$ (13)
Coincidence Failure Diversity	$CFD = \begin{cases} 0 & p_0 = 1.0; \\ \frac{1}{1-p_0}\sum_{i=1}^{R}\frac{R-i}{R-1}p_i & p_0 < 1 \end{cases}$ (14)

The entropy measure (E) ([4, 25])

Given a collection of R classifiers, if l_j denotes the number of classifiers in the collection that correctly recognize x_j, the value of E is then given by Eq. (6). A value of 0 for E signifies no difference between classifiers while a value of 1 indicates the highest possible diversity.

Kohavi-Wolpert measure (KW) ([22, 25])

The Kohavi-Wolpert measure of diversity [22] is given in Eq. (7). [25] goes on to observe that the KW value can be calculated by multiplying the average disagreement measurement (Dis_avg) by a coefficient (Eq. (8)).

The measure of interrater agreement (κ) ([8, 10, 25])

This metric measures the level of agreement between the classifiers. If $\overline{p}$ is the average individual classification accuracy (i.e. $\overline{p} = \frac{1}{NR}\sum_{j=1}^{N}\sum_{i=1}^{R} m_{ji}$), the value of κ can then be defined using Eq. (9). The value of κ can also be obtained through the use of the KW value or the average disagreement measurement, as observed in [25] (Eq. (10)).

The measure of difficulty (θ) ([18, 25])

This measure was first introduced in an article by Hansen and Salamon ([18]. Let A be a random variable that takes values in $\{\frac{0}{R}, \frac{1}{R} \ldots, 1\}$ and denotes the proportion of classifiers in the collection C that correctly classify an input pattern x drawn randomly from the input data set X. The R classifiers in C are run on the data set X to obtain the probability mass function of A. If the collection of classifiers is diverse, a small variance of A is obtained. Collections of similar classifiers are characterized by high variance. The metric 'difficulty' (θ) is defined as the variance of A, i.e. $Var(A)$.

Generalized diversity measure ([25, 33])

This methodology was first introduced in [24]. Let define another random variable B that denotes the proportion of classifiers that are incorrect on a randomly selected input pattern ($B = 1 - A$). And let p_i be the probability of B taking the value $\frac{i}{R}$, and $p(i)$ the probability that i randomly chosen classifiers fail on a randomly chosen input pattern x. Then, using Eqs (11) and (12), we can define the generalized diversity (GD, Eq. (13)). $GD = 0$ signifies minimum diversity, while $GD = 1$ signifies maximum possible diversity.

Coincidence failure diversity measure ([24, 25])

This measure was also introduced in [24]. This function returns a maximum value of 1 when all misclassifications are unique, otherwise a value between 0 and 1. The basic definition of this function is given in Eq. (14).

3.2.3 Other diversity measures

[28] uses the definition presented in Appendix to provide an alternative framework to the measurement of diversity within multi-classifier systems. Given c output classes denoted by $\pi = \{w_1, \ldots, w_c\}$, N inputs denoted by $X = \{x_1, \ldots, x_N\}$ and R classifiers denoted by $C = \{C_1, \ldots, C_R\}$, the classifier outputs for a given input x_n are regarded as a population – $C(x_n) = \{C_1(x_n), \ldots, C_R(x_n)\}$. Then, the within-population diversity with respect to that particular input pattern, $H(P)$, is given by the entropy of the distribution of class labels among classifiers. [28] proposes the use of the $Gini$ [3] index for this measurement. If P_i

[3] The Gini coefficient is a measure of inequality developed by the Italian statistician Corrado Gini and published in his 1912 paper "Variabilit e mutabilit". It is usually used to measure income inequality, but can be used to measure any form of uneven distribution. The Gini

is the probability that a randomly chosen classifier outputs label w_i, where obviously $\sum_{i=1}^{c} P_i = 1$, the Gini diversity among classifiers is defined by Eq. (15). The average diversity among the classifiers over the entire data set X is obtained through averaging G over the entire data set.

$$H(P) = G = 1 - \sum_{i=1}^{c} P_i^2 \tag{15}$$

Another alternative approach presented in [28] is to instead use the populations of output values generated by each classifier when all input patterns are presented. Given two such populations produced by classifiers C_i and C_j, the total diversity $H(P_i, P_j)$ is given by Eq. (16) and can be considered equivalent to the *measure of disagreement Dis*. Considering binary-output classifiers, i.e. '0' or '1' that represents correct or wrong classifications, respectively, the two distributions can be mixed in an output space using 4 elements (11, 10, 01, 00). Then, let denote the probabilities for the joint outputs as $P(11) = a$, $P(10) = b$, $P(01) = c$, $P(00) = d$, and let choose the typical distance, ζ (see Appendix), i.e. $\zeta(m, n) = 1$ iff $m \neq n$, and 0 otherwise. The diversity measure (disagreement expectation) is denoted by Eq. (16).

$$\begin{aligned} H(P_i, P_j) &= (\zeta(1,1) \times a) + (\zeta(1,0) \times b) \\ &\quad +(\zeta(0,1) \times c) + (\zeta(0,0) \times d) \\ &= Dis \end{aligned} \tag{16}$$

4 MCS topology

[46] claims that the combination strategy used to combine the classifiers is of secondary importance, and that the major factor contributing to the success of a multi-classifier system is the diversity among the classifier team. Nevertheless, [26] claims that the choice of an appropriate fusion strategy can further improve the performance of the ensemble. This claim can be substantiated by considering a multi-classifier as a committee of experts. Such a committee reaches its final decision by virtue of voting. This would neglect the individual specialities of the members. Thus, such a strategy would seem pointless if the constitution of the committee is not properly set up. A good solution to such a problem would involve a strategy that would assign areas of expertise to different team members and then follow the relevant experts advice when new items are discussed. Such a strategy would be further enhanced by having each member express a degree of confidence. Problems could be encountered when the expert concerned makes a mistake.

Such problems could be detected by correctly evaluating the committee. Sample problems could be provided to the committee and the results could be analyzed in order to obtain an optimal procedure for the combination of the decision. In terms of multi-classifiers design, this step is the training phase. Another solution to such problems would be the construction of a proper data-flow path before any decision is taken. Decisions could be first made by a subset

coefficient is a number between 0 and 1, where 0 corresponds with perfect equality (where everyone has the same income) and 1 corresponds with perfect inequality (where one person has all the income, and everyone else has zero income). The Gini index is the Gini coefficient expressed in percentage form, and is equal to the Gini coefficient multiplied by 100. (http://en.wikipedia.org/wiki/Gini_index).

of the committee members, and their conclusions are then sent to another subset of members. The construction of such multiple-decision making steps within a decision making system is analogous to the topological construction of a multi-classifier. The proper construction of this topology is usually required to improve the accuracy of the entire system and to incorporate error checks.

[31] categorizes MCS topologies into three categories: Cascading, Parallel and Hierarchical (Fig. 2). In a cascading classifier, the classification result generated by a classifier is used as an input into the next classifier. The results obtained through each classifier are similarly passed onto the next classifier until a result is obtained through the final classifier in the chain. The main disadvantage of the use of this methodology is the inability of later classifiers to correct mistakes made by earlier classifiers.

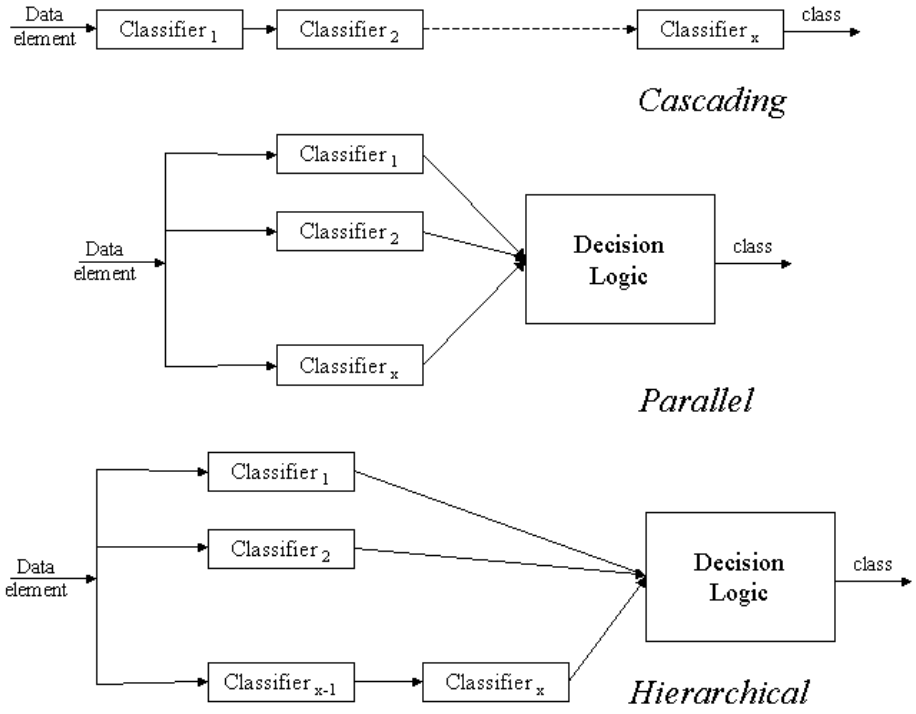

Figure 2: Configurations for combining classifiers

Parallel classifiers integrate the results of each classifier in a singular location. The main design decision that has to be made in the implementation of such a configuration is the selection of a representative combination methodology. If the decision process is well designed, the system can reach peak performance. Some of the more popular and successful combinatorial methods include majority voting, belief integration and the 'stacking until convergence' method. However, the improper selection of a combinatorial strategy could accentuate the influence of poorly performing classifiers, which in turn could adversely affect the overall

performance.

Hierarchical classifiers combine both parallel and cascading classifier configurations to obtain optimal performance. The use of such a methodology can negate the disadvantages encountered through the use of a cascading integration. Hierarchical systems could also be used to introduce error checking, which would nullify the influence of poorly performing classifiers.

A more comprehensive and topical categorization of multi-classifier topologies is presented in [29]. This categorization divides the MCS topologies into conditional, hierarchical, hybrid and multiple-parallel topologies.

- *Conditional topology*: This strategy first selects one classifier to perform the task of classification. If this classifier fails to correctly identify the presented data, another classifier is selected. Most implementations include a primary classifier, which is usually selected as the first classifier to be employed. The selection of the next classifier can either be a static decision or may be based on the values obtained by the use of the primary classifier. Example methods for dynamic selection include decision trees. This process can continue for as long as there are classifiers available or the pattern is correctly classified. If the primary classifier is an efficient one, the process would then become computationally efficient. The queue of selected classifiers could be placed such that the computationally heavy classifiers are only selected at the end of the classifier queue. One difficult aspect of such an implementation is the selection of a process by which the failures and successes of a classifier are judged. This method can become overly complicated when the number of classifiers available increases.

- *Hierarchical (Serial) topology*: This topology employs a method where classifiers are applied in succession. Each classifier applied is used to reduce the number of possible classes to which the input data belongs to. As the data passes through the classifiers, the decision becomes more and more focused. The common way for the design of the classifier queue is to insert classifiers in decreasing values of the error. That is, the classifier with the highest error is used first, whereas the classifier with the lowest error is used last. Of course, there should be safeguards to ensure that the classes selected by any classifier should always include the correct class. If not, the next classifier will not have the option of selecting the correct output class.

- *Hybrid topology*: A hybrid topology based system should incorporate a mechanism for the selection of the best classifier for a given input. This selected classifier is then used to perform the final classification of the given input. It is obvious that certain classifiers perform better than others on certain data. Thus, the selection of an appropriate classifier would streamline the entire classification process.

- *Multiple (Parallel) topology*: This is the most common implementation of a multi-classifier system. All the classifiers first operate in parallel on the input, and the results are then pooled to obtain a consensus result. This methodology does incur a cost as it is computationally heavy, with each classifier having to be executed before the final result is obtained.

The hierarchical and multiple topologies are also known as *selection-based* and *fusion-based* topologies, respectively. The classification presented by [29] is more topical and relevant than the one presented by [31], due to the consideration of hybrid systems. Hybrid intelligent systems are rapidly gaining popularity among researchers due to the limitations of each individual machine learning methodology (these limitations are introduced in Section 1).

Data dependent classification is also rapidly gaining popularity among MCS designers. Such systems involve the creation of multiple combinatorial methods which are specialized towards different points within the output space. The decision on which combinatorial function to use is made based on the features of the input data. Therefore, classifier and combiner selection is dependent on the features present within the input data. This topology is further explained in Section 6. A data dependent classifier presents the closest equivalent to the committee based classifier presented at the beginning of this section. If a decision which is to be made is going to affect the accounting of the organization, it makes sense that the company's accountant gets a larger input to the decision than the others within the committee. Similarly, an advertising decision would be more biased towards the marketing manager. Such intelligent combiner selection which places conditional bias towards individual classifiers within the classifier ensemble makes data dependent classifier fusion an extremely effective classifier fusion methodology.

5 Combinatorial functions

Once the individual classifiers have been designed and implemented, the next most important task involves the combination of the individual results obtained through each individual classifier. With the use of multiple learning paradigms, one obtains a set of classifiers $C_1, C_2, \ldots, C_R$, each of which attempts to approximate a function $f(x)$. A multiple classifier attempts to combine these results in order to obtain a better approximation of $f(x)$. That is, it attempts to combine the outputs $f_{C_1}(x), f_{C_2}(x), \ldots, f_{C_R}(x)$ using the combinatorial function $f_C(x)$.

[51] categorizes multiple classifier combiners into 2 broad categories: feature-vector-based methods[4] and syntactic-and-structural methods.[5] [5] provides a more informative categorization by classifying them as linear, non-linear, statistical and computationally intelligent. They can be described as follows:

- *Linear combination methods*: The use of linear functions like 'SUM' and 'PROD' for the combination of the outputs.
- *Non-linear combination methods*: The use of rank based classifiers like majority voting.
- *Statistical*: The use of methods like the Dempster-Shafer technique and Bayesian combination.
- *Computationally intelligent (CI)*: The use of intelligent paradigms like neural networks and genetic algotithms for combination.

[4] e.g. neural networks.
[5] e.g. fuzzy rule bases.

[5] further categories these approaches according to the existence of a degree-of-confidence for each individual classifier, as confidence based and non-confidence based. Confidence based classifiers are composed of classifiers which have different degrees of certainty associated with them. The confidence is expressed as a weight or an importance which is associated with the output of each classifier before combination within the multi-classifier system. These confidence measures are reflections of the importance of each of the classifiers. However, non-confidence based methods are based on the premise that confidence associated with each classifier within a combination is identical. That is, the degree of importance associated with each classifier is the same. A non-confidence based combiner would be analogous to a company board where decisions are made with each member of the board having an equal vote. A board where decision making bias is allocated according to the type of decision to be made is analogous to a confidence based combiner.

A more refined categorization is offered by [9], where he also categorizes multi-classifier fusion strategies into two categories: fixed combiners and trained combiners. Fixed combiners do not require any form of training and are defined by the designer of the system. These combiners directly use the outputs of the individual classifiers, either as class labels or confidences, to make their final decision. On the other hand, trained classifiers require a learning phase similar to that described in Section 1. The rules for combination are not determined by the designer. We use this classification for the presentation of multi-classifier issues within this paper.

Section 5.1 lists out a collection of the most basic and easily implementable combinatorial functions. Section 5.2 discusses the implementation of these paradigms within a fixed combination methodology and Section 5.3 discusses implementations within a trained combiner.

5.1 Common combinatorial functions

Consequent to the details listed out in the preceding section, we list below the most popular base combiners currently utilized in MCS design.

MAX – maximum. Along with the 'MIN' combinatorial function, this is the simplest implementation of a combiner. Here, the classifier output with the highest value or confidence is chosen as the output of the overall combiner. If the outputs of the classifiers are denoted by $C_j(x)$, where $j = 1, \ldots, R$, then the output of the MAX combinatorial function is given by Eq. (17). However, this method can prove to be inefficient when skewed data sets are used. If the data is adversely either positive or negative dependent, the combiner can also become correspondingly dependent. This combiner can also become very dependent on confident classifiers and can totally isolate other classifiers. The 'MAX' functions works best with measurement level classifier outputs, but can be affected negatively by a single classifier which is performing badly. If the measurements outputted are formatted in a descending manner, the classifier becomes very sensitive to overly predictive classifiers. Similarly, if the measurements are formatted in an ascending manner, the MCS becomes overly protective in that it will not produce maximal classifications without every single classifier agreeing on

the decision.

$$\bar{l}MCS_{MAX}(x) = \max\{C_1(x), \ldots, C_R(x)\} \tag{17}$$

MIN – minimum. Identical to the 'MAX' function, except for the fact that it selects the minimum output value as the combiner output. The 'MIN' combiner has the same advantages and disadvantages as those listed for the 'MAX' combiner, except for the fact that the MCS becomes overly sensitive for ascending formats and overly protective for descending formats.

SUM – summation. The basic form of the 'SUM' operation (i.e. using a non-confidence based approach) involves the use of the summation function to add the outputs of all the classifiers within an ensemble. A more advanced version of the SUM operation involves the assignment of weights for each classifier, and the summation is performed by first multiplying the output of each classifier with its assigned weight and then adding all the obtained values. More exactly, the output of the sum combinatorial function is generated using Eq. (18). Within the basic version we have $w_j = 1$ for all $j = 1, \ldots, R$. Conversely, the value of w_j can vary within the advanced version, depending on the importance or significance assigned to the classifier concerned. These two implementations of the SUM functions can be categorized as being non-confidence-based and confidence-based, respectively.

$$MCS_{SUM}(x) = \sum_{j=1}^{R} C_j(x) w_j \tag{18}$$

PROD – product. This method uses an algorithm similar to the 'SUM' method. However, instead of summing up the weighted outputs of the individual classifiers, it multiples the values. The assignment of values for the weights depends on whether a confidence based or non-confidence based method is used. The basic output of the function can be summarized as $\prod_{j=1}^{R} C_j(x) w_j$. The weights assigned to each classifier can either be equal or dependent on the bias placed on the individual classifier. A more refined definition of this function, for measurement level classifiers, is given in Eq. (19). Here c represents the number of output classes and $C_{ji}(x)$ is the belief that classifier C_j has of input x belonging to class i.

$$MCS_{PROD}(x) = \mathbf{arg}\max_{i=1}^{c} \prod_{j=1}^{R} C_{ji}(x) \tag{19}$$

AVG – average/median. The average method uses a paradigm similar to that used by the 'SUM' method. However, the final output of the combinatorial function is the value of the summation divided by the number of classifiers considered. A formal definition of this combiner is presented in Eq. 20. This function can also be implemented in a confidence based or non-confidence based fashion, by assigning representative weights to each classifier. The AVG combiner works best when the outputs of the classifiers are on a measurement level. The proper assignment of confidence

based weights could be used effectively to nullify the effects of poorly performing classifiers and to enhance the influence of adequately performing ones.

$$MCS_{AVG}(x) = \frac{\sum_{j=1}^{R} C_j(x) w_j}{R} \tag{20}$$

Average vote. Each classifier within the MCS outputs an array which contains numbers between 0 and 1 representing its belief on the compatibility of the given input pattern with each output class (i.e. at a measurement level). The combinatorial function adds the votes for each output class and select the class with the highest vote as the winner. The formal definition of this combinatorial function is given in Eq. (21). Here, $C_{ji}(x)$ represents the vote that classifier C_j makes with respect to input x belonging to class i.

$$\begin{aligned} &MCS_{AVGVOTE}(x) = \\ &\mathbf{arg}\max_{i=1}^{c} \left(\tfrac{1}{R} \sum_{j=1}^{R} C_{ji}(x) \right) \end{aligned} \tag{21}$$

Weighted average vote. The weighted average method is similar to the average vote method. The only difference is that each classifier is assigned a weight which is associated with its output. The weights usually represent an extra confidence or significance that the combiner assigns to each classifier. The definition of this combinatorial function is provided in Eq. (22).

$$\begin{aligned} &MCS_{W-AVGVOTE}(x) = \\ &\mathbf{arg}\max_{i=1}^{c} \left(\tfrac{1}{R} \sum_{j=1}^{R} w_j C_{ji}(x) \right) \end{aligned} \tag{22}$$

BORDA. The Borda count is a voting technique which was developed by Jean-Charles de Borda in 1770. The classifiers used within a Borda count system should be rank-based classifiers (see Section 1). The Borda count is a voting method in which the voter ranks the candidates according to their preference. Each ranking gives each candidate (output class) a certain number of points, with the winner getting the highest number of points. Then, the Borda count functions adds the number of points assigned to each candidate and picks the winner.

MV – Majority voting. The majority voting method is the simplest of all combinatorial functions. It selects the relevant class by polling all the classifiers to see which class is the most popular. Whichever class gets the highest vote is selected. This method is particularly successful when the classifiers involved output binary votes. A case study of the use of this technique is presented in Section 7.

Bayesian combination. Bayesian combiners are used to linearly combine the probabilistic predictions of multiple models by weighing their posterior probabilities ([14]. The basic Bayesian combiner model effectively uses the prior classification probability of the classifiers, $p(j)$ for the j^{th} classifier. For an input x, computations on the marginal likelihood, $p(x|j)$, for each j, $j = 1, \ldots, R$, are performed. Then, using Bayes rule, the posterior probability can be calculated as $p(j|x) = p(x|j)p(j)/p(x)$. These posteriors

can then be used to calculate the classifiers predictions using the formula given in Eq. (23).

$$\begin{aligned} p(w_i|x) &= \sum_{j=1}^{R} p(w_i, C_j|x) \\ &= \sum_{j=1}^{R} p(w_i|x, C_i)p(C_i) \end{aligned} \tag{23}$$

[14] goes on to state that this approach suffers from three limitations. Firstly, that the method is only valid if all the classifiers each capture mutually exclusive and exhaustive possibilities on how the data were generated. Secondly, the difficulties involved in calculating a marginal likelihood. Finally, the fact that all classifiers do not usually start with the same prior assumptions. To rectify these limitations, [14] proposes a Bayesian combiner which does not assume any of the classifiers to be the best and do not expect the classifiers to be probabilistic.

The Dempster-Shafer approach. The Dempster-Shafer (D-S) theory of evidence was introduced in [43]. The D-S theory is used for the representation of uncertain knowledge. This theory was adapted for multiple-classifier combination in [51].

The combiners presented within the preceding list are only a collection of the most common combiners available. The entire list of combiners used is however much larger. Table 3 compares the advantages and disadvantages of each of the functions described above.

[21] claims that all ensemble combination methods can be derived out of the basic product and sum combinatorial methods. Let the numerical outcome or measurement vector of each classifier for input x be denoted by $C_j(x), j = 1, \ldots, R$. Let there exist c output classes, each represented by $w_i, i = 1, \ldots, c$. Here, each class is modelled by the probability density function $p(C_j(x) \mid w_i)$, where the prior[6] probability of occurrence is denoted by $P(w_i)$.

Then, given $C_j(x)$, the input pattern x should be assigned to class w_k if the aposteriori[7] probability of the interpretation is maximized. This condition is listed out in Eq. (24). For simplifying the notations we will get rid of x in the following equations, so we will use C_j instead of $C_j(x)$.

$$\begin{aligned} &P(w_k \mid C_1(x), \ldots, C_R(x)) \\ &= \max_i P(w_i \mid C_1(x), \ldots, C_R(x)) \end{aligned} \tag{24}$$

By using Bayes theorem, [21] obtains Eq. (25). Here, $p(C_1(x), \ldots, C_R(x))$ is the unconditional measurement of the joint probability distribution and can be expressed using Eq. (26). As this value is the same as the first value of the numerator of Eq. (25) , we can concentrate on the denominator. Equation (25) also states that in order to utilize all the available information correctly to reach a decision, it is essential to compute the probabilities of the various hypotheses

[6] A probability calculated by logically examining existing information.

[7] Reasoning which proceeds from effects to causes; which deduces general rules by looking at patterns is called a posteriori reasoning. A statement whose truth can only be found out by experience is also called a posteriori.

Table 3: Advantages and disadvantages of different combinatorial functions

Combinatorial function	Advantages	Disadvantages
MAX & MIN	Works well on either measurement level or rank based classifiers.	Need to assume that the classifiers are conditionally statistically independent. Need to also assume that the a posterior probabilities calculated by the classifiers do not deviate from the prior probabilities. Performance can be adversely affected by either a skewed data set or a poorly performing classifier
SUM	Can be used as either a confidence-based or non-confidence based. Through the use of confidence based approaches, the adverse effects of poorly performing classifiers can be nullified.	Need to assume that the classifiers are conditionally statistically independent. Need to also assume that the a posteriori probabilities calculated by the classifiers do not deviate from the prior probabilities.
PROD	Provides a more averaged result than the SUM combiner. It is less susceptible to skewed data sets than the SUM combiner.	Need to assume that the classifiers are conditionally statistically independent. Susceptible to poorly performing classifiers affecting overall performance.
AVG	Can be used as confidence based or non-confidence-based and consequently, overtrained and undertrained classifiers can be adequately weighted to nullify extra sensitiveness.	Sensitive to classifier bias
BORDA	Easiest combiner to implement for rank-based classifiers.	The combiner loses information on classifier accuracy due to the use of ranks. Very similar inputs could be classified incorrectly due to only the rank of belief being used.
MV	The simplest method to implement. Can be implemented in a confidence-based manner.	Need to assume that the classifiers are conditionally statistically independent. Need to also assume that the a posteriori probabilities calculated by the classifiers do not deviate from the prior probabilities.

by considering all the measurements simultaneously. [21] then uses these values to derive formal definition for both the 'SUM' and 'PROD' combiners. These

Table 4: Probabilities used for evaluation of fixed combining rules [9]

Name	Value	Description
Local Accuracy	$\eta_j(x) = \max_i\{P_i(x)\}$	The probability that x is correctly classified by classifier j.
Expected Accuracy	$\eta_j = E_x[\max_i\{P_{ij}(x)\}]$	The expected local accuracy of classifier C_j.
The Estimate of Local Accuracy	$\hat{\eta}_j(x) = \max_i\{C_{ij}(x)\}$	The estimate of $\eta_j(x)$. Here, it is assumed that the classifier outputs $C_{ij}(x)$ are estimates of the confidences $P_{ij}(x)$.
Accuracy on Evaluation set	$\hat{\eta}_j = \sum_k max_i\{C_{ij}(x_k)\}$	An evaluation set is used for finding the expected accuracy is usually the training set.

two definitions are in turn used to derive formal definitions for the MAX, MIN, AVG and majority voting combiners. For more details, please refer to [21].

$$\bar{l}P(w_i \mid C_1(x), \ldots, C_R(x)) = \frac{p(C_1(x), \ldots, C_R(x) \mid w_i)P(w_i)}{p(C_1(x), \ldots, C_R(x))} \tag{25}$$

$$\bar{l}p(C_1(x), \ldots, C_R(x)) = \sum_{k=1}^{c} p(C_1(x), \ldots, C_R(x) \mid w_k)P(w_k) \tag{26}$$

In addition to the base combiners mentioned above, there are numerous other combiners available through the use of intelligent and statistical techniques. Statistical techniques involve the use of posterior class probabilities to allocate weights for each classifier. Example implementations include the use of Bayes networks and Hidden Markov Models. Intelligent combiners use intelligent learning paradigms for the proper combination of the classifier outputs. However, this has a major drawback as it requires additional training and testing data for training the combiner. This could prove problematic when domains with few training samples are considered. Neural networks have proven to be the most popular intelligent ensemble combiner. The outputs of the individual classifiers can be used as inputs into a neural network. In contrast to base combiners, neural networks were shown to be very effective for the combination of classifiers which are not diverse enough. But, of course, they are also highly effective when adequately diverse classifiers are used. The case study presented in Section 7 compares the use of this method with the use of the other base combiners discussed in this section.

5.2 Fixed combiners

Fixed combiners customarily perform well when the individual classifiers demonstrate similar accuracies and have none or negative correlation between their outputs ([38]). [38] defines such classifiers as '*balanced*' classifiers.

[9] uses posterior class probabilities to evaluate classifiers and lists different fixed combination methods available in the literature. Most of them were presented in the previous section. Let denote the confidence that pattern x belongs to class w_i, where $i = 1, \ldots, c$, as $P_i(x) = P(w_i \mid x)$. Classification consists in assign the pattern to the class with the highest confidence. Also, C_{ij} is defined as the numerical output of classifier j for class w_i, as used before in this paper. Thus, the confidence that classifier j indicates the object x belonging to class w_i is defined as $P_{ij}(x) = P(w_i \mid C_{ij}(x))$. These values are calculated on the whole training set. Then, the values in Table 4 can be computed. A well trained classifier should produce $\hat{\eta_j} \approx \eta_j$, whereas an over-trained classifier will produce $\hat{\eta_j} \gg \eta_j$.

The fixed combiners listed in the previous section can be successfully used to improve the overall performance of the multi-classifier system. This is particularly the case when the feature spaces of the individual classifiers are different. Nevertheless, there have also been cases of improved performance when the different classifiers were trained on the same feature space [9]. But, [9] concludes that these methods only work well under strict conditions. The ensemble will be sub-optimal if the individual classifiers generate unreliable confidences, but will work well if the training set is large and if the combination function is chosen well.

5.3 Trained combiners

These methodologies are in contrast to the fixed combination methodologies listed out in the above section, as they adapt the combiner to the classification task using a training set. The following is a list of guidelines that should be taken into consideration when constructing a classifier of this nature [9]:

1. *Calibration of the individual classifiers*: Each classifier should be trained individually and their outputs should be scaled to within a uniform range for integration.

2. *Selection and weighting of the individual classifiers*: The individual classifiers can differ from each other in terms of performance, which can be measured using a standard evaluation set. The results obtained through this test can be used for the selection of classifiers or for the selection of the weights which will be assigned to the individual classifiers within the combination (this is called global selection). The training set can also be used to partition the feature space into regions, and these regions can then be used to identify the best classifiers for each region. The latter is called local selection of clasifiers.

3. *Global selection of combination methods*: In addition to the selection of the individual classifiers to be used within the ensemble and the weights to assign to each classifier, a selection of the best combination strategy has to be made.

[9] argues that trained combiners perform better than fixed combiners. They conclude that, in fixed combination based systems, confidences are treated according to their interpretation, but the combination rules are sub-optimal. Trained combinations may be assimptotically optimal instead. It is emphasized that the proper training of the individual classifiers is essential.

5.3.1 Combination of neural network classifiers

Of the different methodologies used for the combination of multiple neural net classifiers, majority voting, neural networks, Bayesian inference and the Dempster-Shafer theory have proven the most popular ([32, 51, 52], see Section 5). Although the Dempster-Schafer method has proven to be successful, it is considerably dependendent on the function used for the alignment of the probabilities. For the type of output produced by neural networks, posterior class-conditional probabilities can be calculated. The calculation of these probabilities becomes relatively simple, especially when the number of output classes is small. The case study presented in Section 7 uses neural networks as individual classifiers. Therefore, all the combinatorial functions tested out in this section are implemented to combine neural network classifiers. We also introduce a novel improvement of the standard majority voting algorithm which performs ensemble learning on neural network outputs using a genetic algorithm.

5.3.2 Hybrid combiners

The use of two or more intelligent techniques in conjunction is known as a 'hybrid intelligent' techniques. Most intelligent learning paradigms require the optimization of its internal parameters for proper classification. A neural network requires the definition of the number of hidden layers, the number of neurons within each layer, a learning rate, etc. A weighted average combiner requires the determination of combiner weights. Such optimization challenges provide ideal opportunities for the use of evolutionary computing paradigms. Paradigms such as genetic algorithms, particle swarm optimization and the k-nearest neighbor technique have successfully been used for such implementations.

The use of random forests ([2]) for classifier combination has also gained popularity. A random forest is a collection of classification trees, each of which is dependent on a random variable sampled independently. Classifier selection is performed on the generated solutions using predefined selection criteria. An example implementation of random forest on MCS can be found in [48], where a random forest is utilized for the modeling of structural relationships between molecules.

[34] presents a framework, MultiNNProm, which utilizes a genetic algorithm for the optimization of parameters within a variation of the LOP combiner presented in [17].

6 Data dependent classifier selection

A MCS performs a mapping from the input domain space into the output domain space. This mapping could either be linear or non-linear. The sum rule and average rule (both weighted and non-weighted) are examples of a linear mapping. The maximum and product rules are examples of non-linear mappings.

All MCS outputs are inherently dependent on the input data. However, the attributes which describe the data could also be used to select an appropriate combination (or fusion) strategy. Multiple fusion strategies could be constructed, each of them specialized towards the recognition of a certain class or set of classes belonging to the input space. The use of a multiple fusion strategy of this manner is known as data dependent classifier fusion or combination. The output could

be independent of the input data or implicitly dependent on the input data. An alternative to this is an explicit dependence on the input data. In the last case, the MCSs rely on the characteristics of the input data for the selection of a combination strategy. This classification of classifier fusion strategies, which is detailed below, was presented in [50].

6.1 Data independent strategies

These systems exclusively rely on the output of the individual classifiers. No other information is used in performing classification. As an example, the general model used for the implementation of the MAX function in such systems is shown in Eq. (27), where M_i is a vector which contains the predictions made by all classifiers within the ensemble for the membership of pattern x to class w_i. $M_i(x)$ could be considered equivalent to $C_{ij}(x)$, $\forall j = 1, .., R$, if the previous notation is to be used. The combinatorial mapping $F(\bullet)$ can either be linear or non-linear.

$$MCS(x) = \mathbf{arg}\max_i(F(M_i(x), \forall i), \tag{27}$$

Most of the voting systems, like average, maximum and majority votings are examples of data independent methodologies.

6.2 Implicitly dependent on the data

The combiners in such multi-classifier systems are trained to maximize the global performance on the data. The basic model used for the implementation of the MAX function is defined in Eq. (28), where $W(C(x))$ is a weight matrix (of size cXR) representing the relevance assigned to each classifier within the ensemble, and $C(x)$ represents the vector containing the outputs (or confidences) of all classifiers for input x. Within W, w_{ij} represents the weight assigned to classifier j with respect to class i. Here, the weights are dependent on the outputs of the individual classifiers for a given input. Example implementations of such a methodology include the weighted average ([19]) and fuzzy composition ([13]) fusion strategies. The disadvantage of this approach is that it is vulnerable to consistent mistakes of certain individual classifiers.

$$MCS(x) = \mathbf{arg}\max_i(F(W(C(x)), M_i(x)), \forall i) \tag{28}$$

6.3 Explicitly dependent on the data

In this case, the selection of a classifier or the selection of a combinatorial method is dependent on the sub-space to which the input pattern belongs to. Thus, the final classification is dependent on the initial partition of the input space. The base model used for the implementation of the MAX function is defined in Eq. (29). Here, the weight matrix, W, is dependent on the input pattern, and not on the outputs of the classifiers. Such methods are also called *dynamic classifier selection* (DCS) methods. In dynamic selection methods, the selection of the relevances assigned to individual classifiers is primarily based on the input patterns. A kind of dynamic classifier selection based MCSs which are rapidly gaining popularity among researchers are known as feature-based architectures.

$$MCS(x) = \mathbf{arg}\max_i(F(W(x), M_i(x)), \forall i) \tag{29}$$

[16] provides a framework for the creation of a DCS based MCS which uses the behavior of the MCS as the selection criteria for classifier selection. [15] uses the results obtained in [16] to outline a general purpose theoretical framework for the dynamic selection of classifiers and also proposes two new methods for the implementation of a DCS based system. [55] uses DCS for the mining of noisy data streams. [39] uses the k-nearest neighbor algorithm to calculate local accuracies for the dynamic selection of classifiers.

A system could also be composed of a hierarchical mixture of experts, where different combiners are trained to specialize on different aspects of the input space. [20] uses such a mixture of experts along with an Expectation-Maximization (EM) algorithm for the selection of experts to analyze robot dynamics.

6.4 Feature based architectures

Feature based architectures are used to construct dynamically adapting multi-classifiers. The input pattern can be used to either select classifiers for combination, the combinatorial function or both. Therefore, the system adapts itself to the behavior of the classifiers and the state of the input.

A feature based classifier uses a set of generators to deduce weights for each classifier selected. These weights would represent the confidence that the generator would have on the relevant classifier to correctly classify a given input pattern.

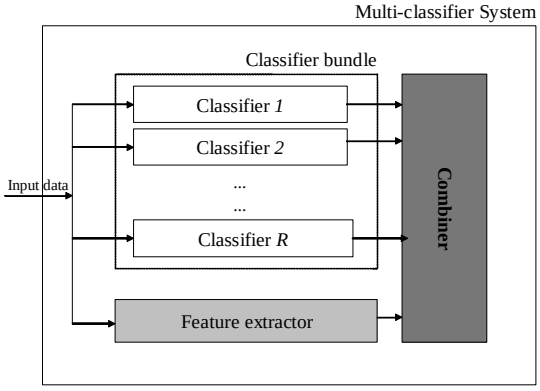

Figure 3: Feature based classification based on input values [50].

Two architectures for feature based systems are proposed in [50]. These two architectures differ from each other by the criterion by which classification is performed. The first architecture only considers the values of the inputs in order to perform feature extraction, whereas the second architecture considers both the input values and the outputs of the individual classifiers for feature extraction. The first architecture is depicted in Fig. 3. Within this model, each classifier C_j produces an output which represents its classification of x. The

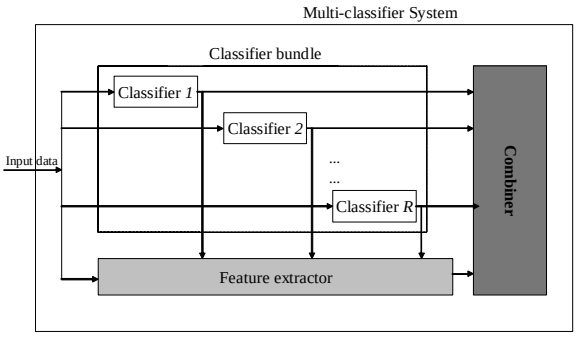

Figure 4: Feature based classification based on both the input values and the outputs of the individual classifiers [50].

collection of classifier outputs for membership of input pattern x in class w_i is denoted by $M_i(x)$.

The feature extractor, E, is a classifier which extracts meaningful information from the input pattern x. This classifier is not responsible for performing the final classification, but is used to provide a probability, $e_i(x)$, which represents its confidence that input x belongs to class w_i. The extractor outputs, $e_i(x), i = 1, \ldots, c$, are represented by the vector $E(x)$.

The combiner is then used to combine the results obtained from the feature extractor with the outputs of the individual classifiers. This combination could be done via a selection of a representative combination function or by the assignment of representative weights to each classifier output. The output of the entire multi-classifier is represented by Eq. (30). Such a classification is similar to the assignment of specialized tasks to relevant members within a committee. For example, when financial decisions are to be made, emphasis on the decision making process is assigned to the Financial Director and his team.

$$MCS(x) = \mathbf{arg}\max_i(F(E(x), M_i(x)), \forall i) \tag{30}$$

The second feature-based architecture is depicted in Fig. 4. This architecture is similar to that presented in Fig. 3 except in the fact that it also uses the outputs of the individual classifiers to interpret its confidence in each classifier. The output of such a multi-classifier is depicted in Eq. (31). The weight matrix is composed of values of the form w_{ij}, where $i = 1, \ldots, c$ and $j = 1, \ldots, R$. The dependence of the weights on both the individual classifier outputs and the input pattern is represented by $W(x, M_i(x))$. Such an architecture is similar to a committee first listening to the views of each committee member at a decision making stage and, then, assigning relevances to each member based on the quality of the views expressed and the type of the decision to be made.

$$MCS(x) = \mathbf{arg}\max_i(F(W(x, M_i(x)), (\mathbf{M})_i(x)), \forall i) \tag{31}$$

Table 5: Standard performance metrics

Name	Abr	Definition	Description (%)
Sensitivity	Sn	$\frac{TP}{TP+FN}$	The Sn measures the proportion of positive patterns being correctly recognized as being positive.
Specificity	Sp	$\frac{TN}{TN+FP}$	The Sp measures the proportion of negative patterns being correctly recognized as being negative
Positive Predictive Value	PPV	$\frac{TP}{TP+FP}$	The PPV provides a measurement of the percentage of true positives to the total number of patterns recognized as being positive. It measures the probability of a positively predicted pattern actually being positive
Negative Predictive Value	NPV	$\frac{TN}{FN+TN}$	The NPV provides a measurement of the percentage of true negatives to the total number of patterns recognized as being negative. It measures the probability of a negatively predicted pattern actually being negative
Precision	P	$\frac{TP+TN}{TP+TN+FN+FP}$	P measures the percentage of samples correctly classified.
Correlation Coefficient	CC	$\frac{[(TP)(TN)-(FP)(FN)]}{\sqrt{(N_P)(TP+FP)(N_N)(TN+FN)}}$	

7 Case study

This section presents a case-study of the implementation of a few of the methods described in the previous sections. The results obtained are used to provide guidelines for combiner selection. We also provide guidelines for training when the data set is imbalanced in terms of the total number of positive samples and negative samples.

7.1 Performance metrics

The effectiveness of a classifier with respect to a given data set is frequently measured using the Specificity and Sensitivity of the system when tested on a test data set. If a given test data set contains N patterns, let the number of positive and negative samples within the data set be denoted by N_P and N_N, respectively. Let the number of positives correctly recognized as being positive be denoted by TP (True Positives) and the number of positives incorrectly recognized as being negative be denoted by FN(False Negatives). Similarly, let the number of negative patterns correctly recognized as being negative and

Table 6: Cross-validated training of the one-hidden-layer neural network classifiers. Best performers in each training/testing run

Training data set (Positives)	Sp	Sn	PPV	NPV
CDS1	0.88009	0.88343	0.55775	0.9747
configuration	7:28:1	7:35:1	7:28:1	7:10:1
CDS2	0.88773	0.91994	0.59355	0.98342
configuration	7:32:1	7:19:1	7:32:1	7:19:1
CDS3	0.84852	0.94944	0.52964	0.98888
configuration	7:37:1	7:31:1	7:37:1	7:31:1
CDS4	0.81721	0.91713	0.46893	0.98123
configuration	7:37:1	7:28:1	7:37:1	7:28:1

incorrectly recognized as being positive be denoted by TN and FN, respectively. These values can then be used to define the metrics described in Table 5.

The values of Sp and Sn are complementary. The aim of any diagnostic test is to maximize both these values. However, it has been been experimentally and theoretically established that an increase in one value, more often than not, leads to the decreasing of the other. This observation is experimentally proven within the next section. Therefore, the objective of any system designer should be the maximization of the precision (P, see Table 5). However, this average is often optimized by increasing the value of the more dominant of the two values. This dominance is dependent on the number of input patterns, and the number of positives and negatives samples. A data set composed primarily of positive data could yield a good P value with the system performing well on the positives.[8] Such observations confirm the fact that the most desirable state would be maximal value for both Sp and Sn, with the two values being as close to each other as possible. To meet these requirements, we present a novel performance metric, the '*Relationship Index*' (RI). The RI is defined in Eq. (32). Within the RI, a low $|Sp - Sn|$ and a high $(Sp + Sn)$ is most desirable, which in turn leads to a low value of the RI being the most desirable.

$$RI = \frac{|Sp - Sn|}{Sp + Sn} \tag{32}$$

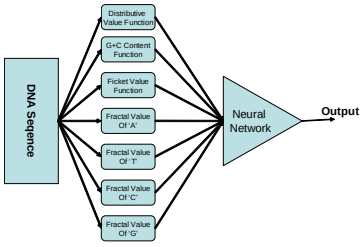

Figure 5: Topological structure of the individual classifiers

[8] i.e. a high Sn value.

Table 7: Cross-validated training of the two-hidden-layer neural network classifiers. Best performers in each training/testing run

Training data set (Positives)	Sp	Sn	PPV	NPV
CDS1	1	0.88624	0.84098	0.97621
Configuration	7:25:1:1	7:11:1:1	7:17:17:1	7:11:1:1
CDS2	1	0.91433	0.84956	0.98258
Configuration	7:27:1:1	7:17:6:1	7:18:8:1	7:17:6:1
CDS3	1	0.95225	0.75212	0.98969
Configuration	7:27:1:1	7:16:19:1	7:10:3:1	7:16:19:1
CDS4	1	0.92416	0.79167	0.98273
Configuration	7:27:1:1	7:23:6:1	7:23:13:1	7:23:6:1
CDS5	1	0.89185	0.86719	0.97796
configuration	7:27:1:1	7:23:12:1	7:25:13:1	7:23:12:1

7.2 Background and classifier construction

In order to prove the effectiveness of an MCS over the use of a singular classifier we created 17 neural network based classifiers. Each of the classifiers was trained to differentiate between coding regions and non-coding regions within DNA strings. We obtained the 'Representative Benchmark Data set of Human DNA' from the Berkeley Drosophila Genome Project website.[9] This dataset consisted of 534 coding regions (CDS regions) and 2946 non-coding regions.[10] Each DNA sequence within the data set was 300 bases long. We divided this data set into 4 sections: two for the training and testing of the individual classifiers and two for the training and testing of the combiner.

Each of the testing/training sets were exercised through 7 property functions to reduce the dimension of the input space. The neural networks were then trained on the output of the 7 property functions. These functions are similar to those used by [49]. The data formatting and training process is graphically illustrated in Fig. 5. Each neural network was trained to output a value of '1' for a correctly recognized coding region and an output of '-1' for a correctly recognized non-coding region.

In order to determine optimal neural network configurations for the produced data set, we first ran a series of neural network training runs to test out all possible 1-hidden neural network configurations. 10 training runs were conducted. Each training run tested out all 1-hidden layer configurations with the number of neurons on the hidden layer varying from 1 to 400. For this series of training runs, the CDS data set was considered as being the positive data set, whereas the INTRON and PROM data sets were used as negative data sets. For the purpose of cross validation, each training run was iterated 5 times, with each iteration using a different data set as the training/testing data set. For example, the first iteration considered CDS1 as the positive data set and used a combination of INTRON1 and PROM1 as the negative training sets. The remaining data were then used to test the accuracy of the trained networks. Each neural network was trained to output a '1' for a recognizes CDS sequence and to output a value of '-1' for a non-CDS sequence. All the neural networks trained

[9] http://www.fruitfly.org/sequence/human-datasets.html.
[10] Introns (INTRON) and promoters (PROM).

Table 8: Performance of selected classifiers for incorporation

Network name	Sp	Sn	PPV	NPV	(Sp+Sn)/2	RI
Net1	0.7974	0.8764	0.4394	0.9727	0.8369	0.047197993
Net2	0.8024	0.8371	0.4344	0.9645	0.81975	0.021164989
Net3	0.834	0.8315	0.4759	0.9647	0.83275	0.001501051
Net4	0.7841	0.8483	0.416	0.9661	0.8162	0.039328596
Net5	0.8035	0.8652	0.4438	0.9705	0.83435	0.036974891
Net6	0.8198	0.8764	0.4685	0.9734	0.8481	0.033368707
Net7	0.8136	0.8427	0.4505	0.9661	0.82815	0.017569281
Net8	0.8126	0.8202	0.4424	0.9614	0.8164	0.004654581
Net9	0.7902	0.8483	0.423	0.9664	0.81925	0.035459262
Net10	0.943	0.7528	0.7053	0.9546	0.8479	0.112159453
Net11	0.8605	0.8258	0.5176	0.9646	0.84315	0.020577596
Net12	0.9511	0.7978	0.7474	0.9629	0.87445	0.087655097
Net13	0.9491	0.7697	0.7326	0.9579	0.8594	0.104375145
Net14	0.8544	0.8427	0.5119	0.9677	0.84855	0.006894113
Net15	0.9613	0.7753	0.7841	0.9593	0.8683	0.107105839
Net16	0.943	0.8202	0.7228	0.9666	0.8816	0.069646098
Net17	0.8625	0.8034	0.5144	0.9603	0.83295	0.035476319

performed very well on the training set and displayed accuracies in excess of 95% on them.

When the testing set was presented to the classifiers, we observed that very few of the sequences were recognized as expected. The outputs of the classifiers were values in the range $[-1, 1]$. Therefore, a hard-limiting(HL) function was utilized for the division of the output space. The initial implementation of the classifiers was executed with a HL of 0.1. The summary of the best performing networks in each training run is provided in Table 6. Each of these networks used 0.1 as their HL.

An identical set of training runs was executed in order to test the difference in accuracy when the number of hidden layers was increased. 10 identical runs training runs were performed with cross-validation. Each cross-validation run tested networks with the number of first hidden layer neurons ranging from 1 to 40 and the number of second hidden layer neurons ranging from 1 to 20. Table 7 lists out the best performing neural networks on each cross-validation training/testing run. Each run provided us with at least one network which performed perfectly on the positives (i.e. with $Sp = 1$). However, each of these networks were extremely negative heavy as they only exhibited Sn values around 0.15 to 0.20.

Of the hundreds of classifiers created, we randomly selected 17. 9 of these contained 1 hidden layer, whereas the other 8 contained 2 hidden layers. Details of the networks selected are given in Table 8.

The accuracies of the individual classifiers could be changed by varying the HL. In order to ascertain the optimal HL for each of the classifiers, we executed all real numbers within the range [−1,1] as HLs for the classifiers. The optimal HLs for the selected classifiers, along with the resulting accuracies are presented in Table 9. Net14 provides the best overall accuracy with the lowest RI value.

Table 9: Optimal performance of the selected classifiers

	HL	Sp	Sn	PPV	NPV	(Sp+Sn)/2	RI
Net1	0.05	0.8632	0.83895	0.52644	0.96729	0.851075	0.014246688
Net2	0.15	0.85845	0.87266	0.52775	0.97382	0.865555	0.008208606
Net3	0.1	0.91887	0.85768	0.6571	0.97269	0.888275	0.034443162
Net4	0.05	0.83876	0.89513	0.50157	0.97784	0.866945	0.032510713
Net5	0.15	0.87101	0.8839	0.55399	0.97641	0.877455	0.007345106
Net6	0.05	0.81127	0.88764	0.46019	0.97551	0.849455	0.044952352
Net7	0.15	0.88206	0.87079	0.59387	0.97442	0.881425	0.012065689
Net8	0.1	0.88715	0.87139	0.61104	0.97853	0.89427	0.003220504
Net9	0.15	0.85064	0.87266	0.51435	0.97358	0.86165	0.01277781
Net10	0.2	0.88357	0.87266	0.57602	0.97454	0.878115	0.00621217
Net11	0.2	0.87916	0.84457	0.55886	0.96895	0.861865	0.020066948
Net12	0.15	0.84555	0.8839	0.50917	0.97571	0.864725	0.02217468
Net13	0.2	0.87814	0.85019	0.55843	0.97	0.864165	0.01617168
Net14	0.2	0.87339	0.87266	0.55542	0.97425	0.873025	0.000418087
Net15	0.2	0.88817	0.87266	0.60836	0.97494	0.885415	0.014405674
Net16	0.2	0.88579	0.85581	0.59817	0.97165	0.8758	0.022824846
Net17	0.2	0.86999	0.8839	0.55205	0.97638	0.876945	0.007930942

Net3 performs best on the negative data set and Net4 performs best on the positive data set. The variance of each of the base metrics with the variance of the HL for one of the selected neural networks is presented in Fig. 6. The complementary nature of Sp against Sn and PPV against NPV is visualized within this figure.

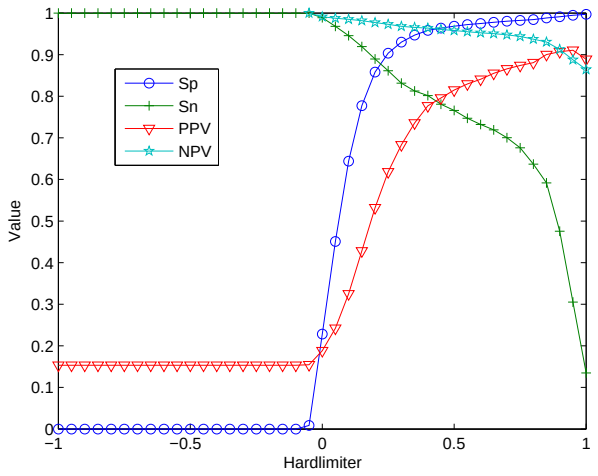

Figure 6: The variance of the performance metrics in relation to the variation of the HL

Table 10: Comparison of final accuracies for the different combiners tested

	Sp	Sn	$(Sp+Sn)/2$	$\lvert Sp-Sn\rvert$	RI
Max	0.85811	0.88951	0.87381	0.0314	0.017967293
Min	0.8852	0.8095	0.84735	0.0843	0.049743317
Mean	0.8985	0.8708	0.88465	0.0277	0.015655909
Median	0.8802	0.8722	0.8762	0.008	0.0183
Standard MV	0.7909	0.9176	0.85425	0.1267	0.074158619
Mean with specialized weights	0.8802	0.882	0.8811	0.0018	0.00102145
MVGen	0.88391	0.8839	0.883905	1E-05	5.65672E-06
MVGen with selection	0.88391	0.8839	0.883905	1E-05	5.65672E-06

7.3 Combination

Figure 7 illustrates the methodology used for the inclusion of the 17 created classifiers within a parallel topology based combiner. Table 10 lists the results obtained through the use of different combinatorial functions for the fusion of the 17 classifier outputs.

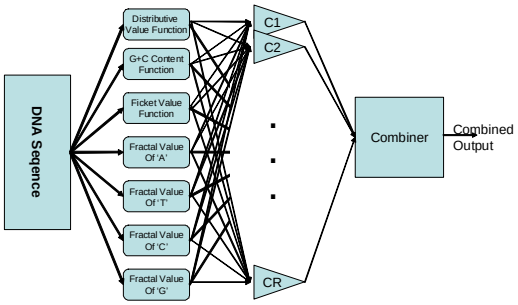

Figure 7: Topological structure of the Multi-Classifier System

The best performing classifier in terms of uniform Sp and Sn was $Net14$, with a Sn of 0.87266 and a Sp of 0.87339. The use of the MAX combiner resulted in an improvement to the sensitivity but resulted in a reduction on the specificity. This result is easily explained via the distribution of data and the output format of the individual classifiers. With the use of the 'MAX' combiner and a coding region input, all the combiner requires for a positive output is for at least one of the classifiers to recognize it as being a coding region. This works conversely when a negative data pattern is used. One classifier erroneously classifying the pattern as being positive would compel the combiner to recognize it as being positive. This results in the combiner being less sensitive to negative data. Conversely, the 'MIN' combiner only requires one classifier to recognize a sequence as being a negative for it to classify the sequence as being a negative sequence. Therefore, the 'MIN' combiner becomes adversely sensitive to negative data, which in turn leads the recognition of a larger number of false negatives. This can result in a lower sensitivity. Therefore, the underlying conclusion of these results is that the use of either the 'MAX' or 'MIN' combiner can be

adversely affected by skewed data sets or a badly functioning classifier.

The drawbacks of the 'MAX' and 'MIN' combiner can be somewhat nullified through the use of either the 'MEAN' or 'MEDIAN' combiner. Through the use of the average value, the effect of biased and badly functioning classifiers can be negated by other properly functioning classifiers. This results in a combined result which is an improvement to the performance of the best classifier within the ensemble.

The standard majority voting algorithm is dependent on the individual thresholds used on each classifier. The results presented in Table 10 are those obtained when each classifier was evaluated using a HL of 0.1. However, by varying this HL, we were able to reduce the difference between Sp and Sn to around 0.01.

The final three combinatorial functions presented in Table 10 are hybrid combiners. The 'Mean with specialized weights' attempts to implement a specialized 'MEAN' combiner with each classifier being assigned a specialized 'bias' or 'weight'. Optimal values for these weights were obtained via a genetic algorithm (GA), where the weight representing each classifier was denoted by a real number within a GA chromosome. The genetic algorithm attempted to obtain optimal values for these weights via the reduction of the RI value of the output.

The 'MVGen' algorithm (Majority Voting optimized with Genetic algorithms) is our approach to training multi-classifier systems for optimized majority voting. Full details and analysis of this approach are yet to be published. Here, we only present the results for comparison reasons. A GA is executed to obtain optimal values for the HL representing each classifier. These optimal HL values are then used to obtain the binary vote of each classifier. These votes are then polled within the combiner to obtain the majority vote. As with the weighted mean combiner, the GA used in this experiment attempted to reduce the RI value of the output. The results obtained through the MVGen combiner displays a significant improvement on the performance of the system in terms of the RI value and the value of both Sp and Sn. The final combiner presented in Table 10 is a variation of the MVGen combiner, where a classifier selection flag is also included within the optimization chromosome for the selection of classifiers for combination. This selection flag determines whether a particular classifier is to be used within the combiner or not. The final optimized system obtained through this experiment yielded us with a system which used 11 of the 17 classifiers available and performed as well as the standard MVGen combiner which used all 17 classifiers.

8 Discussion and conclusions

This paper presented a standard roadmap for MCS designers. We identified the key design decisions which have to first be made when designing a MCS and listed out the options available to the designer at each step. We identified that the incorporation of diversity among the classifiers is of paramount importance in order to obtain better classification via a MCS. We also identified topologies available for the creation of MCS ensembles and presented different situations in which each topology can be utilized. This paper also presents a comprehensive list of the most common combinatorial functions and also analyzes the situations in which each of them are most useful. We also experimentally prove

the functioning of the most common of these functions within certain data set constraints.

We also attempted to introduce the use of hybrid combiners within MCSs. Through the use of weighted majority voting, MVGen and MVGen with selection, we experimentally proved the usefulness and extra functionality of such methods when compared to traditional methodologies.

Appendix: A measure of diversity ([28, 35])

Let denote with P a probability measure associated with a population π. [28] citing [35] provides the following definitions.

Let $(\aleph, ß)$ be a measurable space, and $\wp$ be a convex[11] set of probability measures defined on it. A function $H(\bullet)$ mapping $\wp$ onto the real line is a *measure of diversity* if it satisfies the following conditions:

C1 $H(P) \geq 0$, for any $P \in \wp$ and $H(P) = 0$ iff P is degenerate.

C2 H is a concave[12] function of P.

In other words, $H(P)$ is the diversity within the population π on which the probability measure P is defined.Because of the concavity condition, any mixture of two populations will have a higher diversity than the average of the diversities of the two populations. As indicated in [28, 35], $H(P)$ is the averaged distance ($\zeta(X_2, X_2)$) between two randomly chosen individuals in the population π according to the probability measure P (Eq. (33)).

$$H(P) = \int \zeta(X_1, X_2) P(\partial X_1) P(\partial X_2) \tag{33}$$

If the two individuals (X_1 and X_2) are taken from two different populations π_i and π_j, characterized by the probability measures P_i and P_j, then the total diversity is given by Eq. (34). The dissimilarity between the two populations is given by Eq. (35).

$$H(P_i, P_j) = \int \zeta(X_1, X_2) P_i(\partial X_1) P_j(\partial X_2) \tag{34}$$

$$D_{ij} = H(P_i, P_j) - \frac{1}{2}(H(P_i) + H(P_j)) \tag{35}$$

Because H is a concave function, D_{ij} will be positive for any two populations and their associated probability measures. The distance ζ is a function that satisfies the axioms for distance.[13]

References

[1] P. Baldi and S. Brunak, *Bioinformatics – The Machine Learning Approach*, (Vol. 1), MIT Press, 1998.

[11] for any $P_1, P_2 \in \wp$ and for any $t \in [0,1]$, $tP_1 + (1-t)P_2 \in \wp$.
[12] for any $P_1, P_2 \in \wp$ and for any $t \in [0,1]$, $H(tP_1 + (1-t)P_2) \geq tH(P_1) + (1-t)H(P_2)$.
[13] i.e. nonnegativity, symmetry and the triangle inequality.

[2] L. Breiman, Random forests, *Mach. Learn.* **45**(1) (2001), 5–32.

[3] K. Cherkauer, Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks, *Working Notes of the AAAI Workshop on Integrating Multiple Learned Models*, 1996, 15–21.

[4] Padraig Cunningham and John Carney. Diversity versus quality in classification ensembles based on feature selection. In *ECML '00: Proceedings of the 11th European Conference on Machine Learning*, Springer-Verlag, London, UK, 2000, 109–116.

[5] A.M. de P Canuto, *Combining neural networks and fuzzy logic for applications in character recognition*, PhD thesis, University of Kent, 2001.

[6] T.G. Dietterich, Machine-learning research: Four current directions, *The AI Magazine* **18**(4) (1998), 97–136.

[7] T.G. Dietterich, Approximate statistical test for comparing supervised classification learning algorithms, *Neural Computation* **10**(7) (1998), 1895–1923.

[8] T.G. Dietterich, An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization, *Machine Learning* **40**(2) (2000), 139–157.

[9] R.P.W. Duin, The combining classifier: to train or not to train?

[10] J. Fleis, *Statistical Methods for Rates and Proportions*, John Wiley & Sons, 1981.

[11] Y. Freund and R.E. Schapire, Experiments with a new boosting algorithm, In *International Conference on Machine Learning*, 1996, 148–156.

[12] Y. Freund and R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, *J. Comput. Syst. Sci.* **55**(1) (1997), 119–139.

[13] P.D. Gader, M.A. Mohamed and J.M. Keller, Fusion of handwritten word classifiers, *Pattern Recogn. Lett.* **17**(6) (1996), 577–584.

[14] Z. Ghahramani and H. Kim, Bayesian classifier combination, *Gatsby Technical Report*, 2003.

[15] G. Giacinto and F. Roli. Design of effective neural network ensembles for image classification, *Image and Vision Computing Journal* **19**(9–10) (2001), 699–707.

[16] G. Giacinto and F. Roli, Methods for dynamic classifier selection. In *ICIAP '99: Proceedings of the 10th International Conference on Image Analysis and Processing*, IEEE Computer Society, Washington, DC, USA, 1999, 659–664.

[17] J.V. Hansen and A. Krogh, A general method for combining in predictors tested on protein secondary structure prediction. In *Proceedings of Artificial Neural Networks in Medicine and Biology*, Springer-Verlag, May 2000, 259–264.

[18] L.K. Hansen and P. Salamon, Neural network ensembles, *IEEE Trans. Pattern Anal. Mach. Intell.* **12**(10) (1990), 993–1001.

[19] S. Hashem, Optimal linear combinations of neural networks, *Neural Netw.* **10**(4) (1997), 599–614.

[20] M.I. Jordan and R.A. Jacobs, Hierarchical mixtures of experts and the EM algorithm. Technical Report AIM-1440, 1993.

[21] J. Kittler, M. Hatef, R.P.W. Duin and J. Matas, On combining classifiers, *IEEE Trans. Pattern Anal. Mach. Intell.* **20**(3) (1998), 226–239.

[22] R. Kohavi and D.H. Wolpert, Bias plus variance decomposition for zero-one loss functions. in: *Machine Learning: Proceedings of the Thirteenth International Conference*, L. Saitta, ed., Morgan Kaufmann, 1996, pp. 275–283.

[23] T. Kohonen, *Self-organization and associative memory: 3rd edition*, Springer-Verlag New York, Inc., 1989.

[24] D. Partridge and W. Krzanowski, *Software Diversity: Practical Statistics for its Measurement and Exploitation*, in: *Information and Software Technology*, 39 (1997) pp. 707-717.

[25] L.I. Kuncheva and C.J. Whitaker, Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy, *Machine Learning* **51** (2003), 181–207.

[26] L.I. Kuncheva, A theoretical study on six classifier fusion strategies, *IEEE Transactions on PAMI* **24**(2) (2002), 281–286.

[27] L.I. Kuncheva, That elusive diversity in classifier ensembles, *LNCS* **2652** (2003), 1126–1138.

[28] L.I. Kuncheva, That elusive diversity in classifier ensembles, *Lecture Notes in Computer Science* **2652** (January 2003), 1126–1138.

[29] L. Lam, Classifier combinations: Implementations and theoretical issues. In *MCS '00: Proceedings of the First International Workshop on Multiple Classifier Systems*, Springer-Verlag, London, UK, 2000, 77–86.

[30] L.I. Kuncheva, ed., *Diversity in multiple classifier systems*, (Vol. 6), Elsevier, 2005.

[31] Y. Lu, Knowledge integration in a multiple classifier system, *Appl. Intell.* **6**(2) (1996), 75–86.

[32] E.J. Mandler and J. Schurmann, Combining the classification results of independent classifiers based on the dempster/schafer theory of evidence, *Pattern Recognition and Artificial Intelligence* **X** (1988), 381–393.

[33] D. Partridge and W.B. Yates, Engineering multiversion neural-net systems, *Neural Comput.* **8**(4) (1996), 869–893.

[34] R. Ranawana and V. Palade, A neural network based multi-classifier system for gene identification in dna sequences, *Neural Comput. Appl.* **14**(2) (2005), 122–131.

[35] C.R. Rao, Diversity: Its measurement, decomposition, apportionment and analysis, *Sankya: The Indian Journal of Statistics* **Series A(44(1):1–22)** (1982), 1127–1129.

[36] S.K. Riis and A. Krogh, Improving prediction of protein secondary structure using neural networks and multiple sequence alignments, *Journal of Computational Biology* **3** (1996), 163–183.

[37] G. Rogova, Combining the results of several neural network classifiers, *Neural Netw.* **7**(5) (1994), 777–781.

[38] F. Roli and G. Giacinto, *Hybrid Methods in Pattern Recognition*, chapter Design of multiple classifier systems, Worldwide Scientific Publishing, 2002, 199–226.

[39] F. Roli, J. Kittler and T. Windeatt, Dynamic classifier selection by adaptive k-nearest-neighbourhood rule, *Lecture Notes in Computer Science* **0302** (2004), 174–183.

[40] B. Rost and C. Sander, Prediction of protein secondary structure at better than 70% accuracy, *Journal of Molecular Biology* **232**(2) (July 1993), 584–599.

[41] D. Ruta and B. Gabrys, Analysis of the correlation between majority voting error and the diversity measures in multiple classifier systems. In *Proceedings of the 4th International Symposium on Soft Computing*, 2001, 1824–025.

[42] D. Ruta and B. Gabrys, Classifier selection for majority voting. *Special issue of the journal of INFORMATION FUSION on Diversity in Multiple Classifier Systems*, 2004.

[43] G. Shafer, *A mathematical theory of evidence*, Princeton University Press, 1976.

[44] D. Shalak, The sources of increased accuracy for two proposed boosting algorithms. In *Proc. of AAAI-96, Integrating Multiple Learned Models Workshop* 1996.

[45] A.J.C. Sharkey and N.E. Sharkey, Combining diverse neural nets, *Knowl. Eng. Rev.* **12**(3) (1997), 231–247.

[46] A.J. Sharkey and A.J. Sharkey, *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*, Springer-Verlag New York, Inc., 1999.

[47] C.A. Shipp and L.I. Kuncheva, An investigation into how adaboost affects classifier diversity. In *Proc. IPMU 2002*, 2002, 203–208.

[48] V. Svetnik, A. Liaw, C. Tong and T. Wang, Application of breimans random forest to modeling structure-activity relationships of pharmaceutical molecules, *Lecture Notes in Computer Science* **3077** (Jan 2004), 334–343.

[49] E.C. Uberbacher and R.J. Mural, Locating Protein-Coding Regions in Human DNA Sequences by a Multiple Sensor-Neural Network Approach, *PNAS* **88**(24) (1991), 11261–11265.

[50] N. Wanas, *Features-based architecture for decision fusion*, PhD thesis, University of Waterloo - Canada, 2003.

[51] L. Xu, A. Krzyzak and C.Y. Suen, Several methods for combining multiple classifiers and their applications in handwritten character recognition, *IEEE Trans. on System, Man and Cybernetics* **SMC-22**(3) (1992), 418–435.

[52] L. Xu, A. Krzyzak and C.Y. Suen, Associative switch for combining multiple classifiers, *Journal of Artificial Neural Networks* **1**(1) (1994), 77–100.

[53] G. Yule, *On the association of attributes in statistics*, volume A 194, Phil. Trans., 1900, 257–319.

[54] G. Zenobi and P. Cunningham, Using diversity in preparing ensembles of classifiers based on different feature subsets to minimize generalization error, *Lecture Notes in Computer Science* **2167** (2001), 576–587.

[55] X. Zhu, X. Wu and Y. Yang, Dynamic classifier selection for effective mining from noisy data streams. In *ICDM '04: Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04)*, Washington, DC, USA, IEEE Computer Society, 2004, 305–312.