MC3: A Multi-Class Consensus Classification Framework

Tanmoy Chakraborty[1], Des Chandhok[2], and V.S. Subrahmanian[3]

University of Maryland, College Park, USA
{[1]tanchak, [3]vs}@umiacs.umd.edu; [2]dchandho@terpmail.umd.edu

Abstract. In this paper, we propose **MC3**, an ensemble framework for multi-class classification. **MC3** is built on "consensus learning", a novel learning paradigm where each individual base classifier keeps on improving its classification by exploiting the outcomes obtained from other classifiers until a consensus is reached. Based on this idea, we propose two algorithms, **MC3-R** and **MC3-S** that make different trade-offs between quality and runtime. We conduct rigorous experiments comparing **MC3-R** and **MC3-S** with 12 baseline classifiers on 13 different datasets. Our algorithms perform as well or better than the best baseline classifier, achieving on average, a 5.56% performance improvement. Moreover, unlike existing baseline algorithms, our algorithms also improve the performance of individual base classifiers up to 10%. (The code is available at `https://github.com/MC3-code`.)

Keywords: Ensemble learning, consensus, multi-class classification

1 Introduction

Suppose there are multiple experts sitting together. The moderator gives an object (and its features) and asks the experts to predict its true class from a set of predefined classes. In the first round, experts use their individual heuristics to predict the true class. At the end of the round, every expert discloses her prediction, and learns the predictions made by others. If the moderator does not receive a consensus between the experts' predictions, she allows another round of predictions. In the next round, each expert uses the knowledge of others' predictions, and may modify her heuristics to come up with some other class for that object. Similarly at the end of the second round, the moderator again checks for a consensus. The iteration continues until a consensus is achieved; and finally the class obtained at the consensus is assigned to the object. This is the underlying philosophy of our proposed ensemble classification framework **MC3** (**M**ulti-**C**lass **C**onsensus **C**lassification). Experts are like base classifiers and the final prediction is achieved via consensus.

The power of ensemble classification has been widely accepted by the machine learning community [17]. Existing ensemble classifiers such as Bagging [5], Boosting [18] improve predictions of a base classifier by learning from mistakes. In contrast, our ensemble algorithms support learning across multiple base classifiers in order to achieve consensus to not only achieve high prediction accuracy, but also to improve the performance of base classifiers individually. We propose two versions of **MC3**: (i) **MC3-R**, a recursive version of **MC3**, (ii) **MC3-S**, a two-stage single iterative version of **MC3**. Although **MC3-R** is computationally more expensive, it produces better predictions than **MC3-S** (which slightly trades off accuracy for lower runtime).

We conduct experiments on 13 datasets with different properties (w.r.t. size of data and feature set, class distribution etc.). We compare **MC3-R** and **MC3-S** with 12 (7 standalone and 5 ensemble) classifiers, and observe that our algorithms are either as good as the best baseline or sometimes perform even better than that, achieving an average of 5.56% higher accuracy than the best baseline. Although the best baseline varies from one dataset to another, our algorithm is a single algorithm that achieves the best performance across different datasets. Additionally, the performance of individual base classifiers is improved up to 10%. We also suggest how to select the best parameters for our classifiers.

2 Related Work

Ensemble classification has been an active research area in machine learning (see an exhaustive survey in [17]). Due to the abundance of literature in this area, we restrict our discussion to recent work. Classical ensemble classifiers such as Bagging [5], Boosting [18], Stacking [16], Random Forest [6] etc. [19] use reduced versions of training samples to train ensemble classifiers. *BPNNAdaBoost* and *BPNN-Bagging* [21] built on AdaBoost and Bagging are back-propagation neural network models for financial distress prediction. [20] used an Artificial Bee Colony algorithm for selecting the optimal base classifier and meta configuration in stacking. [9] proposed a classifier ensemble particularly for incomplete datasets. [13] used Artificial Neural Networks with Levenberg-Marquardt back propagation as base classifiers for the Rotation Forest ensemble. [10] combined bagging and rank aggregation. [23] proposed an ensemble classification approach based on supervised clustering for credit scoring. [12] designed a new ensemble pruning method which highly reduces the complexity bagging.

The philosophy behind the existing methods is that base classifiers perform well in different segments of the data and make mistakes in other segments. Ensemble methods combine predictions by balancing between quality and diversity. However, the philosophy behind our ensemble classifiers is completely different – we let each base classifier leverage the predictions made by other classifiers and train itself iteratively to come to a consensus. At the end of the iterations, we expect all the base classifiers to produce *exactly the same* prediction for an unknown instance. This in turn not only provides a strong ensemble classification in general, but also improves the performance of individual base classifiers.

Suppose we are given $\mathcal{S}_{tr}$, a set of M_{tr} training instances taken from a domain $\mathcal{D}$. The i^{th} entry of $\mathcal{S}_{tr}$ is represented by $S_{tr}^{i} = (\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional feature vector[1] and y_i is the true class of S_{tr}^{i} chosen from a set $\mathcal{L}$ (where $l = |\mathcal{L}| \geq 2$). We are also given $\mathcal{S}_{ts}$, a similar set of M_{ts} test instances taken from $\mathcal{D}$; however the true classes of the instances $S_{ts}^{j} = (\mathbf{x}_j, y_j) \in \mathcal{S}_{ts}$ are

[1] We use boldface lower case letters for vectors (e.g., $\mathbf{x}$).

unknown, i.e., $y_j = \phi$. It is also known that each unknown instance belongs to only one class in $\mathcal{L}$. The task is to predict the true class of each instance in $\mathcal{S}_{ts}$.
Ensemble Classification: Let $\mathcal{CF} = \{CF_1, ..., CF_M\}$ be a set of base classifiers. Each base classifier CF_j is trained on $\mathcal{S}_{tr}$ to predict a probability distribution over possible classes for an unknown instance $S_i \in \mathcal{S}_{ts}$, i.e., $\mathbf{p}^j(S_i) = \{p^j(L_1|S_i), ..., p^j(L_l|S_i)\}$, based on which L_i^j is assigned to S_i such that $L_i^j = \arg\max_k p^j(L_k|S_i)$. The final class $L_i \in \mathcal{L}$ of S_i is obtained by feeding the output classes/probabilities obtained from all the base classifiers into an ensemble function $\mathcal{E}(L_i^1, ..., L_i^M)$ or $\mathcal{E}(\mathbf{p}^j(S_i), ..., \mathbf{p}^M(S_i))$. The task is to design an appropriate ensemble function to predict the final class of an unknown instance.

3 Multiclass Consensus Classification

We propose two ensemble classifiers. The first classifier, **MC3-R** is a recursive multi-class consensus classifier that achieves consensus by recursively updating each base classifier using the outcomes of other base classifiers. This classifier turns out to be most accurate, although it suffers from high computational complexity. The second classifier, **MC3-S** is a single iteration multi-class consensus classifier that approximates consensus in one iteration. **MC3-S** is much faster than **MC3-R** and is the closest competitor in terms of accuracy. In the rest of the section, we will elaborate these classifiers.

3.1 MC3-R: Recursive MC3

MC3-R (pseudo-code in Algo. 1; see [1] for a schematic diagram) takes the following inputs – training set $\mathcal{S}_{TR}$, test set $\mathcal{S}_{TS}$, a set of M base classifiers $\{\mathcal{CF}\}_{i=1}^{M}$, a number of iterations $Iter$, a subset selection strategy SS that selects a subset of the M base classifiers, a combination function W, and a consensus function $CONS$. It consists of two fundamental steps – achieving consensus and combining predictions of the base classifiers. **MC3-R** trains each base classifier $Iter$ times on the training set separately and selects the best parameter setting. In each iteration, **MC3-R** achieves consensus after ι levels (the value of ι varies across different iterations). Finally, **MC3-R** combines the outputs of all optimal base classifiers using a weighted function W and predicts the final classes of $\mathcal{S}_{TS}$.
Achieving Consensus: In Step 7 of Algo. 1, **MC3-R** invokes a `getConsensus` function which starts by randomly dividing $\mathcal{S}_{TR}$ equally into $\mathcal{S}_{TR_1}$ and $\mathcal{S}_{TR_2}$ (Step 20). It then calls `getMetaFeatures` twice – in the first (*resp.* second) call each CF_i is trained on $\mathcal{S}_{TR_1}$ (*resp.* $\mathcal{S}_{TR_2}$) to predict $\mathcal{S}_{TR_2}$ (*resp.* $\mathcal{S}_{TR_1}$). The predictions of CF_is then become meta-features for $\mathcal{S}_{TR_2}$ (*resp.* $\mathcal{S}_{TR_1}$) and they are augmented with the original features to generate an expanded feature set $\mathcal{S}^e_{TR_2}$ (*resp.* $\mathcal{S}^e_{TR_1}$). In Step 26, if **MC3-R** reaches consensus based on the consensus function $CONS$ (possible definitions are given in Section 4), it returns the fitness error of individual classifiers which will further be used for best parameter selection (Step 11); otherwise a subset of meta-features are selected using SS (possible definitions are given in Section 4) and augmented with the original features of $\mathcal{S}_{TR}$ to obtain an extended set $\mathcal{S}^e_{TR}$ (Step 40). The entire `getConsensus`

Algorithm 1: MC3-R: Recursive MC3

```
Data: Training set $\mathcal{S}_{TR}$, Test set $\mathcal{S}_{TS}$, No. of iterations $Iter$, Set of $M$ base classifiers
      $\{\mathcal{CF}\}_{i=1}^{M}$, Subset selection function $SS$, a combination function $W$, Consensus
      function $CONS$
Result: Prediction of $\mathcal{S}_{TS}$
1  $\mathcal{P}^*(i) = \phi$, $1 \leq i \leq M$            // Stores the optimal parameters of the classifiers
2  $\mathcal{A} = \phi$                                  // Stores the fitness error of the classifiers
3  $j = 1$
4  while $j \leq Iter$ do
5      $\iota = 1$
6      $\mathcal{P}_i^{\iota} = \phi, 1 \leq i \leq M$
7      $\mathcal{A}$=getConsensus ($\mathcal{CF}$, $\mathcal{S}_{TR}$, $\iota$,$j$)
8      $j++$
9  end
   // Select best optimal parameters of base classifiers from multiple iterations
10 for $(i = 1; i \leq M; i++)$ do
11     $\hat{j} = \text{argmax}_{1 \leq j \leq Iter} \mathcal{A}_j^i$, where $\mathcal{A}_j^i \in \mathcal{A}$
12     for $(k = 1; k \leq \iota; k++)$ do
13         $\hat{CF}_i^k$ is constructed using $Parameter^*(i, \hat{j}, k)$, where $Parameter^*(i, \hat{j}, k) \in \mathcal{P}^*(i)$
14     end
15 end
16 Use $\hat{CF}_i^k$ $(1 \leq i \leq M)$ in $k$ different levels (where $1 \leq k \leq \iota$) to predict the classes of $\mathcal{S}_{TS}$
17 Combine the predictions of $\{\hat{CF}_i\}_{i=1}^{M}$ using $W$ to obtain final classes of $\mathcal{S}_{TS}$
18 return Classes of $\mathcal{S}_{TS}$
19 Procedure getConsensus($\mathcal{CF}$, $\mathcal{S}_{TR}$, $\iota$,$j$)
20     Divide $\mathcal{S}_{TR}$ equally into $\mathcal{S}_{TR_1}$ and $\mathcal{S}_{TR_2}$ randomly
       // Obtain $\iota^{th}$ level optimal classifiers
21     $Parameter^*(i, j, \iota) = 0$, $1 \leq i \leq M$ // Parameters of $CF_i$ at $\iota^{th}$ level of $j^{th}$ iter.
22     $\mathcal{S}_{TR_2}^e, \mathcal{CF}^{*\iota} \leftarrow$ getMetaFeatures($\mathcal{CF}$, $\mathcal{S}_{TR_1}$, $\mathcal{S}_{TR_2}$)
23     $\mathcal{S}_{TR_1}^e, \mathcal{CF}^{*\iota} \leftarrow$ getMetaFeatures($\mathcal{CF}$, $\mathcal{S}_{TR_2}$, $\mathcal{S}_{TR_1}$)
24     $Parameter^*(i, j, \iota) =$ Parameters of $CF_i^{*\iota}(j), 1 \leq i \leq M$
25     $\mathcal{P}^*(i) = \mathcal{P}^*(i) \cup Parameter^*(i, j, \iota)$, $1 \leq i \leq M$
26     if $CONS == True$ then
27         $Error_j^i$ = Fitness error of $CF_i^*$ at the end of $j^{th}$ iteration
28         $\mathcal{A}_j^i = 1 - Error_j^i$
29         $\mathcal{A} = \mathcal{A} \cup \mathcal{A}_j^i$
30         return $\mathcal{A}$
31     end
32     else
33         $\mathcal{S} = \mathcal{S}_{TR_1}^e \cup \mathcal{S}_{TR_2}^e$
34         Use $SS$ to select a subset of meta-features from $\mathcal{S}$ and augment it with the original
             feature set of $\mathcal{S}_{TR}$ to get an expanded feature set $\mathcal{S}_{TR}^e$
35         getConsensus($\mathcal{CF}$, $\mathcal{S}_{TR}^e$, $\iota++$)
36     end
37 Procedure getMetaFeatures($\mathcal{CF}$, $\mathcal{S}_{tr}$, $\mathcal{S}_{ts}$)
38     $K$-fold cross-validation of each $CF_i$ on $\mathcal{S}_{Tr}$ to get optimal classifier $CF_i^*$
39     Use each $CF_i^*$ to predict the classes of $\mathcal{S}_{ts}$ and treat them as meta-features
40     Augment the meta-features with the original features and create an expanded feature
         set for $\mathcal{S}_{ts}$ (call it $\mathcal{S}_{ts}^e$)
41     return $\mathcal{S}_{ts}^e$, $\mathcal{CF}^*$
```

is repeated recursively until the consensus is achieved (Step 35).

Combining predictions: Once consensus is achieved, the best parameters for each classifier at each level are selected based on the fitness error in different iterations (Step 13). The optimal classifiers are then used to predict the class of $\mathcal{S}_{ts}$. Finally, the predictions of the classifiers are combined using W (possible definitions are given in Section 4) to generate the final class of $\mathcal{S}_{ts}$ (Step 17).

3.2 MC3-S: Single Iteration MC3

MC3-S (pseudo-code in Algo. 2; see [1] for a schematic diagram) takes the same inputs as **MC3-R** (except the consensus function $CONS$ since it assumes that consensus is achieved after two levels). **MC3-S** starts by randomly dividing $\mathcal{S}_{TR}$ into two equal subsets $\mathcal{S}_{TR_1}$ and $\mathcal{S}_{TR_2}$ (Step 7). Each classifier CF_i considers $\mathcal{S}_{TR_1}$ and uses k-fold cross validation to obtain optimal parameter settings (we refer to each such optimal classifier as CF_i^*) (Step 8). Each CF_i^* is then used to predict the classes of $\mathcal{S}_{TR_2}$ (Step 9).

In the next step, the classes of $\mathcal{S}_{TR_2}$ obtained from optimal classifiers $\mathcal{CF}^*$ are used as meta-features of $\mathcal{S}_{TR_2}$. We then select a subset of meta-features using SS (Step 10) and augment them with the original features of $\mathcal{S}_{TR_2}$ to get an expanded set $\mathcal{S}^e_{TR_2}$ (Step 11). Note that these optimal classifiers $\mathcal{CF}^*$ will be used later for generating new features. After this, we consider each original classifier CF_i and run k-fold cross-validation on $\mathcal{S}^e_{TR_2}$. This step will produce another optimal set of classifiers denoted by $\{\mathcal{CF}^{**}\}_{i=1}^n$ (Step 12). This set of optimal classifiers will be used later for final class prediction of unknown instances.

The above steps (Steps 7-16) are repeated $Iter$ times, and the optimal parameter settings for $\mathcal{CF}^*$ and $\mathcal{CF}^{**}$ are stored into $Parameter^*$ and $Parameter^{**}$, respectively. At the same time, the accuracies of $\mathcal{CF}^{**}$ are stored in *Accuracy*.

Once $Iter$ iterations are completed, we select the best parameter setting for each CF_i^* and CF_i^{**} based on the values stored in *Accuracy*. We call $\hat{\mathcal{CF}}^*$ for feature generation and $\hat{\mathcal{CF}}^{**}$ for class prediction (Step 17-20). Finally, on the test set $\mathcal{S}_{TS}$, the $\hat{\mathcal{CF}}^*$ classifiers are run to generate meta-features (Step 21), and SS is used to select a subset of meta-features (Step 22). $\hat{\mathcal{CF}}^{**}$ are then run to predict the classes (Step 23). The final class of each instance in $\mathcal{S}_{TS}$ is generated by combining the outputs of $\hat{\mathcal{CF}}^{**}$ using W (Step 24).

4 Functions used in MC3-R and MC3-S

Here, we describe some possible definitions of the functions used in our classifiers.
• **Meta-feature Generation:** Experimental evidence from prior research [11, 22, 16] indicates that augmenting the *confidence* of base classifiers in predicting class levels as meta-features is more useful than considering the *predicted classes* directly. Ting and Witten [22] suggested using as meta-features, the probabilities (often used as confidence values) predicted for each possible class by each base classifier, i.e., $\mathbf{p}^j(S_i) = \{p^j(L_1|S_i), ..., p^j(L_l|S_i)\}$, where $j = 1, \ldots, M$ and $L_k \in \mathcal{L}$. We further extend them by augmenting two additional sets of meta-features for each instance S_i and each classifier CF_j: (i) the probability distribution multiplied by the maximum probability: $\hat{\mathbf{p}}^j(S_i) = \mathbf{p}^j(S_i) \times \max_{1\le k\le l} p^j(L_k|S_i)$, (ii) the entropies of the probability distributions: $E_j(S_i) = -\sum_{k=1}^{l} p^j(L_k|S_i) \cdot \log_2 p^j(L_k|S_i)$. Therefore, the total number of meta-features for each instance would become $M(2l+1)$.
• **Subset Selection:** Instead of considering *all* classifiers, we propose to use SS to select a subset of classifiers for meta-feature generation. Our selection strategies are based on two fundamental quantities – *quality* and *diversity*.

Algorithm 2: MC3-S: Single Iteration **MC3**

```
Data: Training set $\mathcal{S}_{TR}$, Test set $\mathcal{S}_{TS}$, No. of iterations $Iter$, Set of $M$ base classifiers
      $\{\mathcal{CF}\}_{i=1}^{M}$, Subset selection function $SS$, a combination function $W$
Result: Prediction of $\mathcal{S}_{TS}$
1  for $(i = 1;\ i \le M;\ i++)$ do
2      $Parameter^*(i, j) = 0,\ 1 \le j \le Iter$
3      $Parameter^{**}(i, j) = 0,\ 1 \le j \le Iter$
4      $Accuracy(i, j) = 0,\ 1 \le j \le Iter$
5  $j = 1$
6  while $j \le Iter$ do
7      Divide $\mathcal{S}_{TR}$ equally into $\mathcal{S}_{TR_1}$ and $\mathcal{S}_{TR_2}$ randomly
8      $K$-fold cross-validation of each $CF_i$ on $\mathcal{S}_{TR_1}$ to get level-1 optimal classifier $CF_i^*$
9      Use each $CF_i^*$ to predict the classes of $\mathcal{S}_{TR_2}$ and consider them as meta-features
10     Use $SS$ to select a subset of features from the set of meta-features
11     Augment the selected subset of meta-features with the original features and create an
         expanded feature set for $\mathcal{S}_{TR_2}$ (call it $\mathcal{S}^e_{TR_2}$)
12     $K$-fold cross validation of each $CF_i$ on $\mathcal{S}^e_{TR_2}$ to get level-2 optimal classifier $CF_i^{**}$
13     $Parameter^*(i, j)$ = Parameters of $CF_i^*$, $1 \le i \le M$
14     $Parameter^{**}(i, j)$ = Parameters of $CF_i^{**}$, $1 \le i \le M$
15     $Accuracy(i, j)$ = Accuracy of $CF_i^{**}$, $1 \le i \le M$
16     $j = j + 1$
   // Best $CF_i^*$ (resp. $CF_i^{**}$) is used for feature generation (resp. final classification)
17 for $(i = 1;\ i \le M;\ i++)$ do
18     $\hat{j} = \text{argmax}_{1 \le j \le Iter}\ Accuracy(i, j)$
19     $\hat{CF}_i^*$ is constructed using $Parameter^*(i, \hat{j})$
20     $\hat{CF}_i^{**}$ is constructed using $Parameter^{**}(i, \hat{j})$
   // Prediction on test set
21 Use each $\hat{CF}_i^*$ to predict classes of $\mathcal{S}_{TS}$ and use them as meta-features
22 Use $SS$ to select a subset of meta-features and augment it with the original feature set of
     $\mathcal{S}_{TS}$ to get an expanded feature set $\mathcal{S}^e_{TS}$
23 Predict the classes of $\mathcal{S}^e_{TS}$ using $\hat{CF}_i^{**}$
24 Combine the predictions of $\{\hat{CF}_i^{**}\}_{i=1}^{M}$ using $W$ to obtain final classes of $\mathcal{S}_{TS}$
25 return Classes of $\mathcal{S}_{TS}$
```

(i) Quality (Q): We measure the quality of each base classifier in terms of – Area under the ROC curve (AUC) (further used to measure the performance of individual classifiers in Section 6).

(ii) Diversity (D): We measure the diversity between the predictions of base classifiers in two ways: (i) *NMI-based measure*: $D_{nmi} = 1 - \frac{1}{M}\sum_{1 \le i,j \le M} NMI(\mathbf{y}_i, \mathbf{y}_j)$, where Normalized Mutual Information (NMI) is a measure of similarity between two results [15] (see [1] for the definition of NMI), (ii) *Entropy-based measure*: $D_E = \frac{1}{|\mathcal{S}_{tr}|}\sum_{s \in \mathcal{S}_{tr}} \frac{1}{M - \frac{M}{2}} min\{l(s), L - l(s)\}$, where $l(s) = \sum_{i=1}^{M} \delta(Y_i(s), c(s))$, and $\delta(a, b) = 1,\ \text{if } a = b, 0$ otherwise.

Greedy Strategy (G): Given the predictions of all base classifiers $\{\mathbf{y}_i\}_{i=1}^{M}$ as inputs, we select a subset by considering a trade-off between quality and diversity, which can be viewed as a multi-objective optimization problem. We choose a subset $S_{\mathcal{CF}}$ that maximizes the objective function:

$$J = \alpha \frac{1}{|S_{\mathcal{CF}}|} \sum_{i=1}^{|S_{\mathcal{CF}}|} Q(\mathbf{y}_i) + (1 - \alpha) D \tag{1}$$

The parameter α controls the trade-off between these two quantities. However, selecting a proper subset is computationally expensive. Therefore, we adopt the following greedy strategy. We start by adding the solution with highest quality and incrementally add solutions one at a time that maximizes J until local maxima is reached. We set 0.5 as the default value of α. We also consider all the features (ALL) and compare the performance of the classifiers with that of greedy strategy (see Section 6, Tables 2(c) and 2(d)).

• **Output Combination:** In the final stage of our proposed classifiers (Step 17 in **MC3-R** and Step 24 in **MC3-S**), the outputs of the optimal base classifiers are aggregated through a function W. We consider two definitions for W.

(i) Majority Voting (MV): For each test instance we assign the class that the majority of base classifiers agree with. Tie breaking is resolved by assigning that class on which the base classifiers have highest confidence.

(ii) Feature-Weighted Linear Combination (FWLC): As opposed to linear stacking where each base classifier is given a weight, here we assign weights to features. Simple linear stacking defines the weighted function as $W(L_k|s) = \sum_{i=1}^{M} w_i p^i(L_k|s)$, for each $L_k \in \mathcal{L}$ and $s \in \mathcal{S}_{tr}$. FWLC instead models the weight w_i as a linear function of features (including d original and $M(2l+1)$ meta-features), i.e., $w_i(s) = \sum_{j=1}^{d+M(2l+1)} v_{ij} f_j(s)$ for learning weights $v_{ij} \in \mathbb{R}$. Then the weighted function yields the following objective function:

$$min_v \sum_{s\in\mathcal{S}_{tr}} (\sum_{i=1}^{M} \sum_{j=1}^{d+M(2l+1)} (v_{ij} f_j(s) p^i(L_k|s)) - 1)^2 \tag{2}$$

The prediction is subtracted from 1 because we assume that the actual class of s is assigned the probability 1. We use linear regression to obtain the optimal weight for each feature.

• **Consensus Function:** **MC3-R** uses $CONS$ to reach a consensus among the base classifiers. Ideally, all the classifiers should predict the same class for an unknown instance at the end (complete consensus). However, in practice it may not be possible, and therefore we stop **MC3-R** once it reaches a certain threshold of consensus. Two possible definition of $CONS$ are as follows:

(i) Binary Consensus (BIN): For each unknown instance, we check if all pairs of classifiers agree with their predictions: $\frac{1}{|\mathcal{S}_{tr}|}\frac{1}{M}\sum_{s\in\mathcal{S}_{ts}}\sum_{\{CF_i,CF_j\}}\delta(Y_i(s),Y_j(s))$, where $\delta(x,y)=1$, if $x=y$, 0, otherwise.

(ii) NMI-based Consensus (NMI): We measure the average similarity between the prediction of two base classifiers using NMI: $\frac{1}{M}\sum_{\{CF_i,CF_j\}} NMI(\mathbf{y}_i,\mathbf{y}_j)$.

The classifier stops once the difference between the values of the consensus function for two consecutive levels falls below a certain threshold (we take it as 0.02). Later we will see in Figure 2(d) that **MC3-R** achieves consensus within 4-5 levels for most of the datasets.

5 Experimental Setup

Datasets: We perform our experiments on a collection of 13 datasets. These datasets are highly diverse (in terms of size, class distribution, feature size) and widely used. A summary of these datasets is shown in Table 1.

Base Classifiers: Seven (standalone) base classifiers are used in this study: (i) **DT**: CART algorithm for decision tree with Gini coefficient, (ii) **NB**: Naive Bayes algorithm with kernel density estimator, (iii) **K-NN**: K-nearest neighbor

Table 1: The datasets (ordered by the size) and their properties: number of instances, number of classes, number of features, probability of the majority class (MAJ), and entropy of the class probability distribution (ENT). We further report the accuracy (AUC) of our classifiers and the best baseline for different datasets. The best baseline varies across datasets (see Section 6 for detailed discussion).

	Properties of the dataset						Accuracy (AUC)		
	Dataset	# instances	# classes	# features	MAJ	ENT	Best Baseline	**MC3-R**	**MC3-S**
Binary	Titanic [2]	2200	2	3	0.68	0.90	0.66 (SVM)	**0.67**	0.66
	Spambase [14]	4597	2	57	0.61	0.96	0.93 (RF)	**0.95**	**0.95**
	Magic [14]	19020	2	11	0.64	0.93	0.55 (RF)	**0.56**	**0.56**
	Creditcard [24]	30000	2	24	0.78	0.76	0.67 (BAG)	**0.70**	0.67
	Adults [14]	45000	2	15	0.75	0.80	0.78 (SGD)	**0.83**	0.80
	Diabetes [14]	100000	2	55	0.54	0.99	0.64 (RP)	**0.65**	**0.65**
	Susy [3]	5000000	2	18	0.52	0.99	**0.77** (BAG)	**0.77**	**0.77**
Multiclass	Iris [14]	150	3	4	0.33	1.58	0.97 (RP)	**0.98**	**0.98**
	Image [14]	2310	7	19	0.14	2.78	**0.98** (BAG)	**0.98**	**0.98**
	Waveform [14]	5000	3	21	0.34	1.58	0.89 (STA)	**0.91**	0.90
	Statlog [14]	6435	6	36	0.24	2.48	0.92 (RP)	**0.95**	0.94
	Letter Recognition [14]	20000	26	16	0.04	4.69	0.49 (BOO)	**0.54**	0.50
	Sensor [14]	58509	11	49	0.09	3.45	0.98 (BOO)	**0.99**	**0.99**

Table 2: Parameter selection for **MC3-R** and **MC3-S** on Creditcard and Waveform datasets (see abbreviations in Section 4). The accuracies are reported in terms of AUC.

(a) **MC3-R** (Creditcard)

		$SS+CONS$		
		$G:D_{nmi}$+BIN(NMI)	$G:D_E$+BIN(NMI)	ALL+BIN(NMI)
W	FWLC	0.64 (0.63)	0.66 (0.65)	0.65 (0.65)
	MV	0.65 (0.66)	0.67 (**0.68**)	0.66 (0.67)

(b) **MC3-R** (Waveform)

		$SS+CONS$		
		$G:D_{nmi}$+BIN(NMI)	$G:D_E$+BIN(NMI)	ALL+BIN(NMI)
W	FWLC	0.76 (0.79)	0.76 (0.78)	0.77 (0.78)
	MV	0.88 (0.88)	0.86 (**0.91**)	0.89 (0.90)

(c) **MC3-S** (Creditcard)

		SS		
		$G:D_{nmi}$	$G:D_E$	ALL
W	FWLC	0.66	0.66	0.67
	MV	0.67	**0.70**	0.68

(d) **MC3-S** (Waveform)

		SS		
		$G:D_{nmi}$	$G:D_E$	ALL
W	FWLC	0.56	0.56	0.76
	MV	0.90	**0.91**	0.89

algorithm, (iv) **LR**: multinomial logistic regression, (v) **SVM**: Support Vector Machine with linear kernel, (vi) **LDA**: supervised latent Dirichlet allocation [7], (vii) **SGD**: stochastic gradient descent classifier [4]. We utilize standard grid search for hyper-parameter optimization. These algorithms are further used later as standalone baseline classifiers to compare with **MC3-R** and **MC3-S**.

Baseline Algorithms: We compare **MC3-R** and **MC3-S** with the standalone classifiers mentioned earlier. We additionally compare them with 5 state-of-the-art ensemble classifiers: (i) **Linear Stacking** (STA): stacking with multi-response linear regression [16], (ii) **Bagging** (BAG): bootstrap aggregation method [5], (iii) **AdaBoost** (BOO): Adaptive Boosting [18], (iv) **Random Forest** (RF): random forest with Gini coefficient [6], and (v) **RP:** a recently proposed random projection ensemble classifier [8]. Thus, in all, we compare our algorithms with 12 classifiers including sophisticated ensembles.

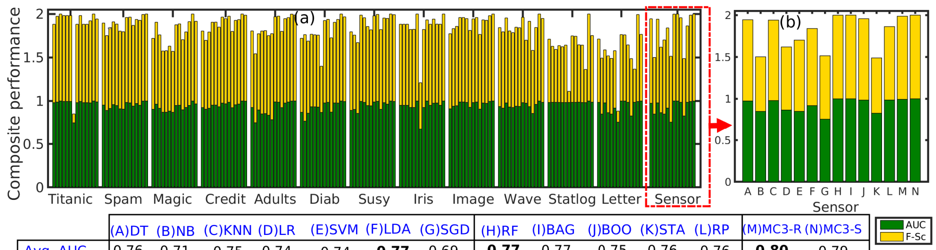
6 Experimental Results

In this section, we first present the parameter selection strategy for our classifiers. In the interest of space, we will only present the results of parameter selection for Creditcard and Waveform (as representatives of binary and multiclass datasets respectively); however exceptions will be explicitly mentioned. Following this, we will present the performance of all the algorithms for different datasets. The performance is reported after 10-fold cross validation. All experiments were performed on a cluster of 64 Xeon 2.4GHz machines with 24GB RAM running RedHat Linux.

Parameter Selection: Table 2 shows the performance of our classifiers for different parameter combinations. For instance, the top left entry in Table 2(a) indicates that the AUC value of **MC3-R** on the Creditcard dataset is 0.64 (*resp.* 0.63) with NMI-based greedy subset selection $G : D_{nmi}$ and binary consensus BIN (*resp.* NMI-based greedy subset selection $G : D_{nmi}$ and NMI-based consensus NMI). We observe that in general **MC3-R** and **MC3-S** perform the best with majority voting (MV) as W (exception including $FWLC$ for the Magic dataset), greedy strategy (G) with entropy-based diversity D_E as SS (exception including greedy D_{nmi}-based strategy for the Sensor dataset) and NMI-based $CONS$. Moreover, for both the classifiers, we observe in Fig. 2(c) that the performance does not change much with the number of iterations ($Iter$); therefore we take $Iter = 1$ to speedup the classifier. The rest of the experiments are conducted with these parameter settings for **MC3-R** and **MC3-S**.

Comparative Analysis: The performance of the classifiers is evaluated based on two evaluation measures – AUC and F-score (see detailed experimental results in the Supplementary Materials [1]). For better visualization, we present here the *composite performance* of all classifiers – for each evaluation measure (AUC and F-score), we separately scale the scores of the competing classifiers so that the best performing classifier has a score of 1. The composite performance of a classifier is the sum of the 2 normalized scores. If a classifier outperforms all others, then its composite performance is 2. Fig. 1 shows that our classifiers outperform others, irrespective of the datasets. The composite performance of **MC3-R** and **MC3-S** is 1.99 and 1.97 respectively, followed by RF (1.92), Bagging (1.92), RP (1.87), DT (1.82), KNN (1.82), BOO (1.92), STA (1.81), LDA (1.81), SVM (1.80), LR (1.80), NB (1.73) and SGD (1.55). The absolute performance of each classifier averaged over all datasets as shown in the bottom table of Fig. 1 indicates that **MC3-R** performs 3.89% (*resp.* 5.56%) better than the best baseline in terms of AUC (*resp.* F-Score). For further comparison, the absolute accuracy of **MC3-R** and **MC3-S** along with the bast baseline is presented in Table 1. Interestingly, we observe in Table 1 that although the best baseline tends to be competitive with our classifiers, there is no particular baseline which is the best across all datasets. Therefore, *one may choose our classifiers rather than spending time deciding on which classifier to choose because our classifiers are at least as good as any existing classifier irrespective of the dataset.*

As expected, we observe in the bottom table of Fig. 1 that **MC3-S** is much faster than **MC3-R**. Note that the runtime is reported after running base classifiers sequentially. We further measure *overhead ratio* of an ensemble algorithm as the ratio between the total runtime of the ensemble algorithm and the total runtime taken by the base algorithms used in the ensemble algorithm. We notice that the overhead ratio of our algorithms is the minimum (around 1; while the maximum is 714 for Bagging; see Supplementary Materials [1]). It essentially in-



		(A)DT	(B)NB	(C)KNN	(D)LR	(E)SVM	(F)LDA	(G)SGD	(H)RF	(I)BAG	(J)BOO	(K)STA	(L)RP	(M)MC3-R	(N)MC3-S
Avg. AUC		0.76	0.71	0.75	0.74	0.74	**0.77**	0.69	**0.77**	0.77	0.75	0.76	0.76	**0.80**	0.79
Avg. F-Score		0.67	0.61	0.67	0.66	0.65	**0.68**	0.59	**0.72**	0.71	0.68	0.66	0.70	**0.76**	0.75
Runtime (Sec.)	Sensor	3	1	365	4	13	2	1	23	3644	161	22581	17622	165573	22872
	Diabetes	7	3	5	4	3	5	22	640	4905	725	6991	39187	313432	102945
	Susy	19	7	2104	26	288	19	38	896	9104	1063	9630	9872	603955	259187

Fig. 1: (a) Composite performance of all the classifiers ((A)-(G): standalone, (H)-(L): ensemble, (M)-(N): ours) on different datasets. (b) The results on Sensor dataset are zoomed out separately. The order of classifiers on x-axis (i.e., labels on x-axis) in (a) is same as that in (b) and is omitted for better visualization. The table below presents the average accuracy of the classifiers over all the datasets, and the runtime on three largest datasets (see [1] for the runtime of all the algorithms on other datasets).

dicates that the runtime of our algorithms is high due to the sequential execution of the base algorithms, which can be parallelized easily.

We further study other aspects of the classifiers:

(i) ***Dependency on the feature size*****:** We consider Spambase[2] having highest number of features (57) and drop 5 features at a time based on descending order of importance[3] and plot AUC in Fig. 2(a). We observe that our algorithms consistently perform well despite dropping features – **MC3-R** almost remains invariant up to 12 features. The reason might be that our classifiers produce additional meta-features to separate instances well in the feature space. This suggests that *our classifiers add high value for datasets with a small number of original features.*

(ii) ***Dependency on the size of the training set*****:** We consider Creditcard[2] and decrease the training size from 75% to 50% (with 5% interval) of the entire dataset. Training set is selected randomly, and for each training size, the average AUC is reported in Fig. 2(b) after repeating it 20 times. We observe that our classifiers are less affected by the training size. Therefore, *one may choose our classifiers when the training size is small.*

(iii) ***Dependency on the number of iterations*****:** In both **MC3-R** and **MC3-S**, we choose the best parameter setting of the base classifiers after running them $Iter$ times. Fig. 2(c) shows that the overall performance does not vary much with an increase in $Iter$. Therefore, we choose $Iter = 1$ to make the classifiers fast.

(iv) ***Convergence of* MC3-R: MC3-R** takes ι levels to achieve consensus. Table 2 shows that NMI-based consensus is more effective than binary consensus. Although there is no theoretical guarantee of achieving consensus since the base classifiers are treated as a black box, we empirically observe that for all the

[2] The patterns are exactly the same for the other datasets.

[3] We separately measure the importance of each feature by dropping it in isolation and calculate the decrease in accuracy (more decrease implies more relevance).

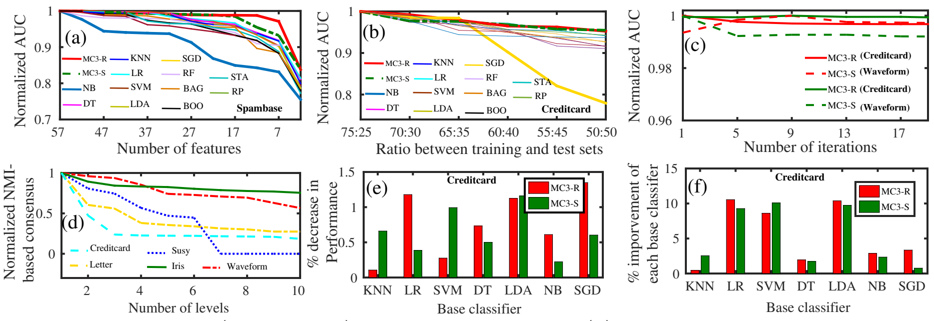


Fig. 2: Performance (normalized) of each classifier for (a) different number of features and (b) different training size. The lines corresponding to **MC3-S** and **MC3-R** are significantly different (McNemar's, $p < 0.005$) from other lines. (c) Performance of our classifiers after considering different number of iterations ($Iter$). (d) Normalized NMI-based consensus of **MC3-R**) over different levels of iterations for different datasets. (e) Decrease in performance of our classifiers after dropping each base classifier in isolation. (f) Performance improvement of each base classifier due to our ensemble classifiers.

datasets **MC3-R** converges after a certain level. Figure 2(d) shows that for small datasets (e.g., Iris) consensus is achieved much faster (within 2-3 levels) than large datasets (e.g., Creditcard, Susy) for which **MC3-R** usually takes 7-8 levels of iterations (on average 4-5 levels for most of the datasets).

(v) ***Dependency on the base classifiers*****:** For each dataset, we drop each base classifier in isolation and measure the change in performance of **MC3-R** and **MC3-S**. Fig. 2(e) shows that LDA affects the performance the most. As mentioned in the comparative analysis, LDA seems to be the best standalone classifier. This may imply that incorporating strong classifiers into the base set may have a bigger impact than incorporating the weak classifiers.

(vi) ***Improvement of individual base classifiers*****:** As opposed to traditional ensemble classifiers, our classifiers improve individual base classifiers separately once consensus is reached. Fig. 2(f) shows the percentage improvement of base classifiers after incorporating meta-features generated by our classifiers. We observe that the improvement is significantly high, ranging up to 10% in some cases. Interestingly, our classifiers are able to gear up the performance of strong base classifiers (such as LDA, LR, SVM) as well.

7 Conclusion

In this paper, we have advanced the paradigm of ensemble classification by providing a new notion of "consensus learning". We have shown that there is no existing classifier which always performs the best across different datasets. Our classifiers are at the top, performing as well or better than the best existing classifier (baseline and ensembles) across all 13 datasets we considered. The rigorous study of 13 different datasets and the comparative analysis with 12 baseline classifiers allows us to assert that achieving consensus not only provides a better way of designing ensemble classifiers, but also enhances the accuracy of individual base classifiers by a significant level (up to 10%).

Acknowledgments. Parts of this work were funded by ONR grants N00014-15-R-BA010 and N00014-16-R-BA01.

References

1. Supplementary Materials. `http://tinyurl.com/PAKDD-MC3`
2. Titanic dataset. `https://www.kaggle.com/c/titanic`, accessed: 2016-09-30
3. Baldi, P., Sadowski, P., Whiteson, D.: Searching for Exotic Particles in High-Energy Physics with Deep Learning. Nature Commun. 5, 4308 (2014)
4. Bottou, L.: Large-Scale Machine Learning with Stochastic Gradient Descent, pp. 177–186. Physica-Verlag HD (2010)
5. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
6. Breiman, L.: Random forests. Mach. Learn. 45(1), 5–32 (Oct 2001)
7. Cai, X., Wang, H., Huang, H., Ding, C.: Simultaneous image classification and annotation via biased random walk on tri-relational graph. In: ECCV. pp. 823–836. Florence, Italy (2012)
8. Cannings, T.I., Samworth, R.J.: Random projection ensemble classification. ArXiv (2015)
9. Conroy, B., Eshelman, L., Potes, C., Xu-Wilson, M.: A dynamic ensemble approach to robust classification in the presence of missing data. Machine Learning 102(3), 443–463 (2016)
10. Datta, S., Pihur, V., Datta, S.: An adaptive optimal ensemble classifier via bagging and rank aggregation with applications to high dimensional data. BMC Bioinformatics 11(1), 427 (2010)
11. Džeroski, S., Ženko, B.: Is combining classifiers with stacking better than selecting the best one? Mach. Learn. 54(3), 255–273 (2004)
12. Guo, L., Boukir, S.: Margin-based ordered aggregation for ensemble pruning. Pattern Recognition Letters 34(6), 603 – 609 (2013)
13. Karabulut, E.M., İbrikçi, T.: Effective diagnosis of coronary artery disease using the rotation forest ensemble method. Journal of Medical Systems 36(5), 3011–3018 (2012)
14. Lichman, M.: UCI repository (2013), `http://archive.ics.uci.edu/ml`
15. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)
16. Reid, S., Grudic, G.: Regularized Linear Models in Stacked Generalization, pp. 112–121. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
17. Rokach, L.: Ensemble-based classifiers. Artif. Intell. Rev. 33(1-2), 1–39 (Feb 2010)
18. Schapire, R.E.: A brief introduction to boosting. In: IJCAI. pp. 1401–1406. Stockholm, Sweden (1999)
19. Schclar, A., Rokach, L., Amit, A.: Ensembles of classifiers based on dimensionality reduction. CoRR abs/1305.4345 (2013)
20. Shunmugapriya, P., Kanmani, S.: Optimization of stacking ensemble configurations through artificial bee colony algorithm. Swarm and Evolutionary Computation 12, 24 – 32 (2013)
21. Sun, J., Liao, B., Li, H.: Adaboost and bagging ensemble approaches with neural network as base learner for financial distress prediction of chinese construction and real estate companies. Recent Patents on Computer Science 6(1), 47–59 (2013)
22. Ting, K.M., Witten, I.H.: Issues in stacked generalization. J. Artif. Intell. Res. (JAIR) 10, 271–289 (1999)
23. Xiao, H., Xiao, Z., Wang, Y.: Ensemble classification based on supervised clustering for credit scoring. Applied Soft Computing 43, 73 – 86 (2016)
24. Yeh, I.C., Lien, C.h.: The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Expert Syst. Appl. 36(2), 2473–2480 (2009)