Mach Learn (2011) 82: 375–397
DOI 10.1007/s10994-010-5224-5

Feature-subspace aggregating: ensembles for stable and unstable learners

Kai Ming Ting · Jonathan R. Wells · Swee Chuan Tan · Shyh Wei Teng · Geoffrey I. Webb

Received: 26 March 2009 / Revised: 1 October 2010 / Accepted: 17 October 2010 /
Published online: 18 November 2010

Abstract This paper introduces a new ensemble approach, Feature-Subspace Aggregating (Feating), which builds local models instead of global models. Feating is a generic ensemble approach that can enhance the predictive performance of both stable and unstable learners. In contrast, most existing ensemble approaches can improve the predictive performance of unstable learners only. Our analysis shows that the new approach reduces the execution time to generate a model in an ensemble through an increased level of localisation in Feating. Our empirical evaluation shows that Feating performs significantly better than Boosting, Random Subspace and Bagging in terms of predictive accuracy, when a stable learner SVM is used as the base learner. The speed up achieved by Feating makes feasible SVM ensembles that would otherwise be infeasible for large data sets. When SVM is the preferred base learner, we show that Feating SVM performs better than Boosting decision trees and Random Forests. We further demonstrate that Feating also substantially reduces the error of another stable learner, k-nearest neighbour, and an unstable learner, decision tree.

Keywords Classifier ensembles · Stable learners · Unstable learners · Model diversity · Local models · Global models

Editor: Mark Craven.

K.M. Ting (✉) · J.R. Wells · S.C. Tan · S.W. Teng
Gippsland School of Information Technology, Monash University, Vic 3842, Australia
e-mail: kaiming.ting@monash.edu

J.R. Wells
e-mail: jonathan.wells@monash.edu

S.C. Tan
e-mail: james.tan@monash.edu

S.W. Teng
e-mail: shyh.wei.teng@monash.edu

G.I. Webb
Clayton School of Information Technology, Monash University, Vic 3800, Australia
e-mail: geoff.webb@monash.edu

1 Introduction

Existing ensemble methods such as Bagging, Random Forests, Random Subspace and Boosting generate multiple **global models** from a single learning algorithm through randomisation (or perturbation) in order to improve predictive accuracy relative to a single model. The reliance on global models and randomisation (or perturbation) also means that only unstable base learners can be used for these ensemble methods because only unstable base learners generate sufficient global model diversity through randomisation or perturbation. This excludes many stable base learners that, when applied directly, may produce more accurate individual models for the given learning task. Unstable learners will generate substantially different models when there is a small perturbation on the training data; whereas stable learners generate models that differ little in the context of small perturbations. Examples of stable learners are Naive Bayes, k-nearest neighbour classifiers and support vector machines. Decision tree learners are a typical example of unstable learners.

This paper introduces a fundamentally different approach to ensemble learning, which induces diverse **local models**, rather than global models, in distinct local regions of the feature-space. The proposed ensemble approach constructs all partitions of the feature-space that are defined over a fixed number of attributes and learns a local model in each partition. This approach can be applied to both stable and unstable learners, and trains faster than existing ensemble methods because each local model is learned using a substantially smaller data set than learning a global model using the whole data set. The reduction in training time will be more pronounced when the learning algorithm has high order polynomial time complexity. In addition, our approach will continue to get improvement by increasing the level of localisation for as long as the data quantity is sufficient to support the level of localisation and the base learner does not produce a global model with Bayes optimal error for the problem.

The proposed work is distinguished from existing work on ensemble methods based on (i) global models that rely on randomisation (such as Bagging (Breiman 1996), Random Forests (Breiman 2001)) or perturbation (Boosting (Schapire and Singer 1999)), feature subset selection/search for ensembles (Opitz 1999; Oza and Tumer 2001; Ho 1998), and (ii) localised modelling (such as NBTree (Kohavi 1996) and Lazy Bayesian Rules (Zheng and Webb 2000)) by:

- Employing a complete enumeration of feature-space subdivisions to build a set of local model variants for each point in the feature-space that ensures no duplicate feature-space subdivisions. In contrast, methods that build global models through randomisation (or perturbation) may contain many replicating local regions in their seemingly different global models;
- Utilising local models in a way that provides more model diversity than existing learners which either heuristically identify or randomly select a small set of feature subsets to build global models (see Sect. 2.3 for more details); and
- Enabling either randomised or enumerated implementation.

We show in this paper that the proposed approach improves the predictive accuracy of a single model and decreases the time required to generate an individual model, as the level of localisation increases. For example, in one of the largest data sets we used, building one model in the proposed ensemble using support vector machines (SVM) takes less than one-hundred-eightieth of the time required to train one single SVM model! The approach works for both stable and unstable models such as decision trees, k-nearest neighbours and SVM.

Our research generalises the approach of Average One-Dependence Estimators (AODE) (Webb et al. 2005) which uses an exhaustive set of one-dependence estimators to form an

ensemble. We show that this strategy is applicable to any base learning algorithm, and extend it beyond the single attribute level of localisation employed by AODE. In other words, AODE is a special case of the approach we propose in this paper. Our research focuses on developing a generic ensemble approach that is parameterised to control the level of localisation and can employ a wide range of learning algorithms to generate the base models for ensembles. The research is also informed by work on random ensembles (Breiman 1996; Liu et al. 2008), recognising the limitations of existing randomisation approaches.

In this paper, we first focus on the stable learner SVM and show that Feating performs significantly better than Boosting, Random Subspace and Bagging in terms of predictive accuracy, and is significantly faster, especially when training a single SVM requires a long time. We also study Feating's performance with respect to increasing level of localisation, and show that it works well with different types of base learners. We then demonstrate that Feating performs better than some common ensemble methods particularly in domains where SVM is a very accurate base learner. Finally, we demonstrate that Feating offers the flexibility to allow the user to switch readily from an unstable base leaner to a stable base learner (or vice-versa) in order to improve predictive performance across different domains.

2 Feature-Subspace Aggregating

A **local model** formed from instances similar to one we wish to classify will often be more accurate than a **global model** formed from all instances (Frank et al. 2003). However, in the general case we do not know the relevant distance metric so do not know what local neighbourhood to use. We propose to use many local distinct neighbourhoods, creating an ensemble by applying the base learner in each.

Our approach is guided by the principle that smaller problems, as a result of Feature-Subspace Aggregating, are easier to solve than a single global problem. It subdivides the feature-space into non-overlapping local regions in a single subdivision; and ensures that different subdivisions provide the distinct local neighbourhoods for each point in the feature-space. There are many ways a feature-space can be subdivided. Instead of using heuristics, we subdivide the feature-space exhaustively based on a user-specified number of features to control the level of localisation. The set of exhaustive feature-space subdivisions forms the basis to develop a new ensemble method which aggregates all local models or a random subset of these local models. We call the proposed method Feature-Subspace Aggregating or Feating.

We describe the enumerated version of Feating in Sect. 2.1, provide the time complexity analysis in Sect. 2.2, and explain the randomised version of Feating in Sect. 2.3.

2.1 Definition and implementation

Let X be the input n-dimensional feature space with A_d denoting the d-th feature, and Y denote the set of classes representing the underlying concept, which is to be learnt from a training set of size m, $D := \{(x_a, y_a) \mid a \in \{1, 2, \ldots, m\}\}$.

Let $e : X \to \{0, 1\}$, and $e(x)$ is a conjunction of h atoms over $A_1, \ldots, A_n$ (or h conditions). The subspace and its complement confined by e are given as follows:

$$r := \{x \in X \mid e(x) = 1\}, \qquad \bar{r} := X \backslash r.$$

We define an h-subdivision as:

$$R = \bigcup_i r_i = X, \quad \text{where } \forall i \neq j,\ r_i \cap r_j = \emptyset,$$

r_i are mutually exclusive subspaces that cover the entire feature space using conjunctions of h features, and $h \leq \frac{n}{2}$. A learning algorithm f is used to build a local model $f(r_i)$ for each subspace r_i.

Let r_i^k be a subspace i of an h-subdivision k. The set of all distinct h-subdivisions, in which no two r subspaces are equivalent or a subset of another between any two h-subdivisions, is defined as follows:

$$Rh := \{R \mid \forall \ell \neq k, r_.^\ell \not\subseteq r_.^k\}, \tag{1}$$

$$|Rh| = C_n^h, \tag{2}$$

where C_n^h is the binomial coefficient.

In other words, for each point in the feature space, there are exactly $|Rh|$ distinct $r_.^k$ subspaces in which a local model is trained in each $r_.^k$. Although these subspaces are distinct, it shall be noted that the local models cannot be guaranteed to be distinct in at least two circumstances: (i) When the data distribution is such that data only exist in $r_.^\ell \cap r_.^k$ for some subspaces; and (ii) there are strong correlations among attributes. Nevertheless, the Rh condition minimises the number of redundant local models in an ensemble.

Feating is an ensemble method which combines all local models generated from Rh or a subset of Rh.

An example of Feating in binary domains In a domain of n binary attributes, the feature-space can be subdivided into two half-spaces n ways, where each such subdivision uses one of the n attributes. Extending to subdivision using h attributes, the feature-space can be subdivided into $\frac{1}{2^h}$-spaces in C_n^h ways, where h is the number of binary attributes used to do the subdivision. In general, an h-subdivision is one of the possible non-overlapping subdivisions of the feature-space using h attributes. The exhaustive set has C_n^h h-subdivisions for a given feature-space defined by n attributes.

Let us consider that a subspace is a local neighbourhood of a point x in the feature space. Applying the subdivision, each point x has exactly C_n^h different local neighbourhoods; and x is the minimum intersection of all the local neighbourhoods. Each subspace is defined by one conjunction of h conditions. For example, $\{A_1 = 0$ and $A_2 = 1\}$ defines a subspace constrained by attributes A_1 and A_2 and their specified values. The exhaustive set of x's local neighbourhoods contains all the different training data sets from which x can be modelled (by using a learner to produce one local model for each local neighbourhood). Models learned from these distinct local neighbourhoods can be viewed as the representative local models for x. For example, if the feature space is defined by four binary attributes A_1, A_2, A_3 and A_4 and $x = \langle A_1 = 0, A_2 = 1, A_3 = 1, A_4 = 0\rangle$; for $h = 2$, six sets of quarter-spaces r_i of two dimensions are shown below, where each column indicates the attributes used (with an implied value for each attribute) in each quarter-space:

A_1	A_1	A_1	A_2	A_2	A_3	Level 1
A_2	A_3	A_4	A_3	A_4	A_4	Level 2

which define all six local neighbourhoods for x. This example shows the enumeration method we employed to generate C_n^h h-subdivisions in a Feating ensemble which can be done without attribute selection. This is implemented as function 'attributeList' shown in Algorithm 1. This requires a ranking of attributes to be done beforehand. We implement this ranking using information gain in the function 'rankAttribute'.

h specifies the level of localisation—a high h signifies a high level of localisation with a reduced training data size in each subdivision; and only attribute subsets, which do not

Algorithm 1: Feating(D, A, h)
– Build a set of Level Trees based on Feating

INPUT D—Training set, A—the set of given attributes, h—the maximum level of Level Tree

OUTPUT E—a collection of Level Trees

$E \leftarrow \emptyset$
$n \leftarrow |A|$
$N \leftarrow C_n^h$
$P \leftarrow$ rankAttribute(A)
for $i = 1$ to N **do**
  /* Construct an attribute list from P based on index i */
  $L \leftarrow$ attributeList(P, i)
  $E \leftarrow E \cup$ BuildLevelTree($D, L, 0$)
end for
Return E

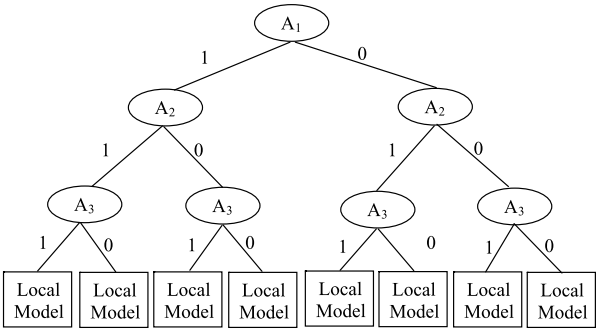

Fig. 1 An example of Level Tree with three levels of localisation using attributes: A_1 at level 1, A_2 at level 2, and A_3 at level 3, and a local model attached to each leaf. Each Level Tree with local models forms a single global model

contain attributes used in r to define a local neighbourhood, are used to build the local model for the neighbourhood.[1]

Level Tree The data structure we propose, which implements the Feature-Subspace Aggregating mentioned above, is called a *Level Tree*. It is a restricted form of decision tree where each node at a single level of the tree must use the same attribute. Figure 1 shows an example of a Level Tree defined by three attributes: A_1, A_2 and A_3. A local model is trained from data in each leaf of the tree and it is attached to the leaf. To classify an example x, one first traverses the tree from the root to a leaf following the branches that x satisfies. Then the model at that leaf is applied to x to produce a classification. A Level Tree[2] with local

[1]This is true if r is using the equality operator for a single value. When r is using an inequality operator (for numeric attributes) or the equality operator for a set of values (for nominal attributes having more than two labels), then local models will be built using all attributes in the subspaces.

[2]As a single model, a Level Tree is similar to an Oblivious Decision Tree (ODT) (Kohavi and Li 1995), except that an ODT uses the majority class of the training instances at a leaf to classify whereas a Level Tree uses local models. However, ODT is meant to be used as a single model with no considerations for ensembles.

models is thus equivalent to a single global model, ready to predict when a test instance is presented.

Using Feature-Subspace Aggregating, the structures of all possible h-subdivision Level Trees (without local models) can be generated with minimal cost by enumeration since no attribute selection is required, unlike in the case of ordinary decision trees. Subdivision for numeric attributes will be considered in the same manner as a cut-point selection in an ordinary decision tree using a heuristic such as information gain. As a result, though each attribute can appear at one level only in a Level Tree, the cut-points used for a numeric attribute on different nodes of the same level can be different.

The training data is filtered through the branches in the Level Tree as each node is created. If the training data is less than some minimum number (n_{min}) before a branch reaches the required level h or all instances belong to the same class, a leaf is formed; and the majority class of instances at the leaf is used for prediction. Otherwise, a local model is trained for each branch that has a level h. The default setting, $n_{min} = 4$, is used in all of our experiments reported in this paper.

Algorithm 2 shows the procedure to build a Level Tree.

The aggregation of all possible Level Trees of h-subdivision forms an ensemble. To make the final prediction, the predictions from all Level Trees in an ensemble are aggregated using a simply majority vote, like in bagging.

2.2 Time complexity analysis

Assume an algorithm with time complexity $O(m^p)$ is used to generate 2^h local models in each binary Level Tree of h-subdivision, where m is the number of training instances and p is a constant. The total execution time of generating a single binary Level Tree is in the order of $2^h m^p$, ignoring the time to generate the structure of a Level Tree which is negligible by comparison in nominal attribute domains.[3] Further assume uniform data distribution and every increment of h reduces the number of instances in each subdivision by half. Thus, the ratio of reduction in execution time (T_R) for each Level Tree as a result of a single increment of h is given as follows:

$$T_R = \frac{2^h m^p}{2^{h+1}(m/2)^p} = 2^{p-1}$$

Compared to generating a single global model, the ratio of reduction in execution time for generating a single model in Feating of h-subdivision is thus $2^{h(p-1)}$. Generalising to b-nary trees, where $b \geq 2$, the ratio of reduction will be $b^{h(p-1)}$. This reduction may be viewed as an upper bound because of the uniform data distribution assumption. In contrast, most ensemble methods employing randomisation have the ratio of reduction approximately equal to one since generating a single model and each of its variants from a base learning algorithm requires about the same amount of time when using the same data size, defined by the same number of features.

Figure 2 shows an example of the ratios of reduction in execution time of a single model using Feating with quad trees (i.e., $b = 4$ because each attribute in this domain has four labels) at $h = 1$, $h = 2$, and $h = 3$; and Feating employs three different base learning algorithms: J48 (decision tree), IBk (k-nearest neighbour where $k = 5$) and SVM which have

[3]Only in the case of numeric attributes, computation is required for cut-point selection for an attribute, predetermined by enumeration, in each Level Tree node—this cost is the same as that in the construction of a node of an ordinary decision tree but minus the cost of attribute selection.

Algorithm 2: BuildLevelTree(D, L, j)
– Build a Level Tree recursively

INPUT D—Training set, L—Attribute set, j—Current tree level

OUTPUT *node*—Level Tree node

```
Global variables:
/* B is the base learning algorithm */
/* h is the maximum level of the Level Tree */
if j = h then
   node.localModel ← BuildLocalModel(D, B)
   Return node
end if
/* n_min—#instances required before a split is allowed. */
if D belongs to the same class or |D| < n_min then
   Return a leaf labelled with the majority class
end if
/* Retrieve next attribute from L based on current level */
v ← nextAttribute(L, j)
Construct a node with attribute v
if v is a numeric attribute then
   node.splitpoint ← findSplitPoint(v, D)
   D_1 ← filter(D, v > node.splitpoint)
   D_2 ← filter(D, v ≤ node.splitpoint)
   node.branch(1) ← BuildLevelTree(D_1, L, j + 1)
   node.branch(2) ← BuildLevelTree(D_2, L, j + 1)
   m ← 2
else
   /* Split based on nominal attributes */
   let {v_1, ..., v_m} be the possible values of v
   for i = 1 to m do
      D_i ← filter(D, v = v_i)
      node.branch(i) ← BuildLevelTree(D_i, L, j + 1)
   end for
end if
if Σ_{i=1}^{m} |D_i| < |D| then
   /* Collect all instances with missing v values to form a new branch */
   D_{m+1} ← filterMissingValue(D, v)
   node.branch(m + 1) ← BuildLevelTree(D_{m+1}, L, j + 1)
end if
Return node
```

different time complexities. We also compare them to Boosting J48 with ensemble sizes equivalent to the three settings of h. The data set (coding) consists of a total of 20000 instances, and the result is averaged over a 10-fold cross-validation. Note that we are applying

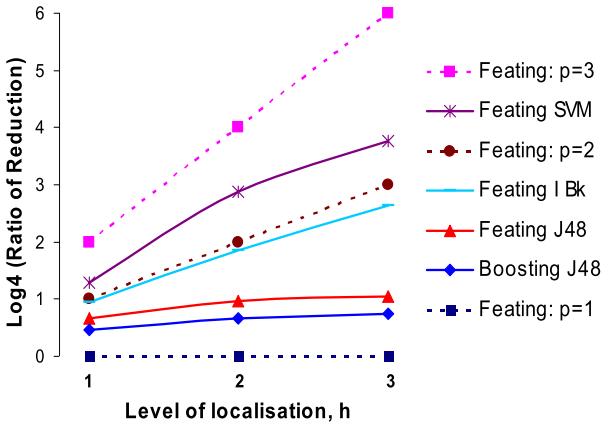

Fig. 2 Plot of ratio of reduction (Log_4) at three levels of localisation for Feating hypothetical algorithms having time complexities $O(m^p)$ (where $p = 1, 2$ & 3), Feating SVM, IBk and J48, and Boosting J48. The numbers of iterations used for Boosting J48 are the same as the numbers of models created by Feating $h = 1, 2, 3$ for the three runs so that they are comparable

$\log_4$ to the reduction ratios and include Feating three hypothetical algorithms with different p values which show three linear lines with different reduction ratios.

The largest reduction comes from Feating SVM which is equivalent to Feating a hypothetical algorithm with a time complexity between $O(m^2)$ and $O(m^3)$; and the reduction ratios for Feating IBk is very close to Feating a hypothetical algorithm with $O(m^2)$. Feating J48 appears to behave as Feating an algorithm with a time complexity between $O(m)$ and $O(m^2)$. The reduction ratio for Boosting is relatively modest and is less than that for Feating J48.

2.3 Randomised version of Feating

Feating can be implemented in more than one way, including a randomised version which randomises the Feature-Subspace Aggregating process (i.e., selecting a random subset of Rh), as opposed to the randomisation/perturbation in the instance space like bagging or boosting, and randomisation in the model building process as in Random Forests. This implementation completely avoids the issue of enumerating all C_n^h models as in Feating without randomisation.

The above randomisation enables feature-subspace subdivisions (i.e., Level Tree structure) to be built without access to the training data; and each Level Tree structure can be constructed very quickly.

Here we show that randomisation can be applied in two parts of the Feature-Subspace Aggregating process. First, Feating randomly selects a subset of h-subdivisions to form an ensemble of a user required size. This ensures that every h-subdivision in an ensemble is unique, without duplication from Rh.

Second, the division at each node of a level tree can be randomised. This can be implemented as follows. For numeric attributes, randomly select a cut-point for division; for nominal attributes having $l > 2$ labels, randomly select a dichotomy (that splits l labels into two non-empty subsets) from a total of C_l^2 dichotomies.

Randomised Feating has two advantages relative to Feating without randomisation. First, the random selection allows Feating to form an ensemble of a smaller size than C_n^h for domains with large n. Second, random divisions at each node of a level tree allows Feating to form an ensemble of a larger size than C_n^h for domains with small n, by generating multiple rounds of C_n^h h-subdivisions where each round consists of random variants of other rounds. Note that multiple rounds of C_n^h h-subdivisions necessitate the relaxation of the condition

in (1) to include h-subdivisions which contain $r_{\cdot}^{\ell} \subset r_{\cdot}^{k}$. In doing so, there is a small chance that $r_{\cdot}^{\ell} = r_{\cdot}^{k}$ between some h-subdivisions in the ensemble.

Randomised Feating, implementing the two ways of randomisation mentioned above, has the following number of violations of the Rh condition. Let e be the required ensemble size, and $d > 1$ be a constant. If $e \leq |Rh|$, then every h-subdivision is unique with no violations. If $|Rh| < e \leq d \times |Rh|$, then the total number of violations of the Rh condition is $e - |Rh|$ and every unique h-subdivision is violated a maximum of $|d - 1|$ times.

Randomised Feating is different from Random Subspace (RS) (Ho 1998) in three key aspects. RS builds an ensemble of global models, each learned from a random feature subset. The first difference is that, RS builds each global model from all data points, albeit with a reduced feature set; whereas Feating builds local models using local training sets defined by subspaces in Rh. Second, the diversity of RS models is confined to the set of C_n^h feature subsets only, assuming h is the number of selected features; whereas the model diversity in Feating is further enhanced by $b^{(h+1)}$ different local models in each of the C_n^h feature subsets, assuming in a b-nary domain. In effect, each RS model is forced to *exclude* a subset of the features (those not selected) and is free to utilise as it sees fit the remaining features to build a global model, whereas each Feating model is forced to *include*, albeit implicitly, a subset of features (those selected to build the level tree) and is left free to utilise those remaining to build local models. Third, for an algorithm with $O(nm^p)$, the training time for RS is expected to be reduced by half only, when $h = n/2$ is used in order to have the maximum diversity; whereas the training time for Feating is expected to be reduced in the order of $\frac{n}{n-h} b^{h(p-1)}$.

Conceptually, Random Forests (Breiman 2001) uses an unrestricted form of tree which means the set of possible subdivisions is a lot larger than Rh, and many of the subdivisions have local regions which are either equivalent or a subset of another—violating the Rh condition. Note that Randomised Feating only violates the Rh condition when $e > |Rh|$; and the number of violations is bounded, as described above. In contrast, it is possible that all trees in a Random Forests ensemble violate the Rh condition, where each tree has at least one subspace which is a subset of a subspace in another tree, i.e., $\forall \ell \; \exists k \neq \ell, r_{\cdot}^{\ell} \subseteq r_{\cdot}^{k}$; and for a fixed ensemble size, the total number of violations in Random Forests can be expected to be substantially larger than that of Feating because the tree structure is unrestricted.

3 Empirical evaluation

We design our experiment to examine the properties of the proposed Feating approach and compare it with Boosting, Random Subspace and Bagging. The experiment is conducted under the WEKA platform (Witten and Frank 2005). We use the k-nearest neighbour, support vector machine, decision tree, Boosting, Random Subspace and Bagging implementations in this platform (i.e., IBk, SMO, J48, AdaBoostM1, RandomSubspace, Bagging). The three base learning algorithms are selected because of their differing characteristics that we want to examine in Feating. They are algorithms which produce different degrees of (un)stable models; have significantly different time complexities; and SMO and J48 have substantially higher training time than testing time, whereas IBk is a lazy algorithm and hence spends most time in the testing process. k is set to 5 for IBk; J48 produces an unpruned decision tree whenever it is used; all other settings are as in the default settings. For Random Subspace, each base model is built using $n/2$ features randomly selected from the feature space. All experiments are carried out on an OpenPOWER 720 Power5 Linux Cluster with 8 gigabyte main memory (www.vpac.org).

Table 1 Data sets used in the experiments. Ensemble sizes for different levels of Feating are also provided. The ensemble size for $h = 1$ is also the number of attributes in each data set

Data Sets	Data Size	Feating's Ensemble Size		
		$h = 1$	$h = 2$	$h = 3$
coding	20,000	15	105	455
nursery	12,960	8	28	56
dna	3,186	60	1,770	34,220
wave40	5,000	40	780	9,880
satimage	6,435	36	630	7,140
wave21	5,000	21	210	1,330
segment	2,310	19	171	969
anneal	898	38	703	8,436

A total of eleven data sets from the UCI Machine Learning Repository (Asuncion and Newman 2007) are used. These data sets are selected because their data sizes and number of features vary substantially. Table 1 provides the characteristics of the first eight data sets used, together with the total number of h-subdivisions at each level of localisation for Feating. Table 7 lists the characteristics of the three data sets in which SVM with quadratic kernel is a more accurate base learner than J48 and SVM with linear kernel. Because none of the base learners (used in this study) can perform consistently well across all these data sets, they are good examples to illustrate how our approach deals with this problem.

All experiments for a fixed ensemble size are conducted using 10-fold cross-validation. When a curve is presented which shows the performance of an ensemble with a range of ensemble sizes, the result for each ensemble size is obtained from one run of train and test with either a 9:1 random split into training and testing sets or the given training and testing sets. We measure the performance in terms of error rate and execution time per model or total execution time for each ensemble. We measure a total of training and testing time in each data set and then divide the total time by the ensemble size to give the average execution time per model in order to demonstrate the time reduction in comparison to that of a single ordinary model. We also use a pair-wise t-test with 5% level of significance to examine whether the difference in error rates is statistically significant between two algorithms.

We first study the performance of Feating without randomisation in the first three sections. In Sects. 3.1 and 3.2, we focus on Feating SVM.[4] We then show that Feating also works well with other stable and unstable learners in Sect. 3.3. In Sects. 3.4 and 3.5, we study the performance of Feating with randomisation. We demonstrate that Feating is the preferred ensemble method in domains where a stable learner such as SVM is a more accurate learner than an unstable learner such as decision trees. Further discussion in Sect. 4.2, we also show that stable and unstable base learners can have different levels of predictive performance across different domains; and an implementation called Feating-*mix* allows the better-performing base learner to excel without choosing explicitly between two different base learners. This is a significant advantage because the effect is achieved without the need to introduce a new mechanism (e.g., to increase model instability) or resort to combining different ensemble methods.

[4]At this stage, our focus is to understand the ensemble performance of Feating, so we make no attempt to change the default settings of SVM in the WEKA platform (Witten and Frank 2005).

3.1 Feating SVM

The aim of this section is to demonstrate that Feating SVM decreases both the error rate and the execution time per model of SVM as the level of localisation h increases.

The error rate and execution time of Feating SVM are presented in Table 2. Averages and geometric means of the performance ratio (using SVM as the reference) over the eight data sets are presented in the last two rows of the table. They show that the average error rates decrease monotonically from 12.79% for SVM to 9.36% for Feating SVM at $h = 3$. The geometric means of performance ratios show that Feating decreases the error rate of SVM by 20% at $h = 1$ and 45% at $h = 3$. There are a few exceptions to the general trend which occur in wave21 and dna. The error rate spike at the highest level of localisation in the dna data set are likely due to the effect of a low number of training examples relative to the high branching factor and high number of attributes. This data set has 60 nominal attributes, each having four labels, and a data size of 3186 examples only. Many branches in the Level Tree have no or a very small number of training examples when $h = 3$. SVM requires more training examples in order to produce a reasonable model; thus it can only tolerate up to $h = 2$. The number of poor performing local models in such branches becomes sufficiently large to affect the performance of the ensemble when h is high. Thus, domains with many multi-label attributes require to either have large data size or use a low level of localisation when employing Feating.

In terms of execution time, the geometric means in Table 2 show that Feating reduces the time to construct and test a model to 27% of SVM at $h = 1$ and 14% at $h = 3$. The general trend persists in every single data set for Feating SVM from SVM to $h = 1$ and $h = 2$. However, there is a slight increase in execution time at $h = 3$ (relative to $h = 2$) in four of the eight data sets. This is likely due to the increase in difficulty (relative to those in the lower levels) in converging to the final SVM local models as a result of small data size in subdivisions at a high h level.

Table 2 Average error rate and execution time (in seconds) for constructing and testing one model of SVM and Feating SVM at $h = 1, 2$ and 3 levels of localisation. Entries with boldface (resp. italic) indicate that the error rates are significantly lower (resp. larger) than those of a single SVM, in a pair-wise t-test with 5% level of significance

	Error rate				Time per model			
	SVM	Feating SVM			SVM	Feating SVM		
		$h = 1$	$h = 2$	$h = 3$		$h = 1$	$h = 2$	$h = 3$
coding	28.58	**26.10**	**23.67**	**20.05**	2392.01	397.45	43.72	13.03
nursery	6.85	**3.75**	**1.57**	**.35**	47.06	7.39	2.79	2.91
dna	7.34	**3.77**	**3.20**	**7.19**	17.02	4.19	2.38	5.14
wave40	13.72	13.68	13.46	13.42	3.60	1.26	1.02	1.10
satimage	13.33	**11.84**	**10.86**	**9.76**	3.51	1.63	1.54	2.19
wave21	12.98	13.00	13.16	*13.42*	2.37	.70	.48	.48
segment	6.97	**6.41**	**5.63**	**4.68**	2.04	.70	.57	.52
anneal	12.58	**9.91**	**7.13**	**6.01**	1.89	.53	.32	.30
Average	12.79	11.06	9.84	9.36	308.69	51.73	6.60	3.21
Geomean		.80	.65	.55		.27	.14	.14

3.2 Feating versus Boosting, Random Subspace and Bagging using SVM

Table 3 compares the error rates of Feating with Boosting, Random Subspace (RS) and Bagging using SVM as the base model. Boosting makes little or no improvement on the predictive performance of SVM, except in one data set (anneal). RS actually increases the error rates relative to that of a single SVM model and becomes significantly worse in five out of eight data sets. It manages to significantly lower the error rate of the single SVM on the dna data set only. Bagging makes no significant improvement on the predictive accuracy of SVM in all but one data set. In contrast, Feating reduces error rates in seven out of eight data sets; the differences are significant in all data sets, except the wave21 and wave40 data sets.

In Table 4, the total time results show that Feating uses significantly less time than Boosting in the three most time consuming data sets (coding, nursery and dna), in which training a single SVM takes a long time. Notably Feating of eight $h = 1$ trees with sixteen local SVM models uses only a slightly more time than a single global SVM model in nursery, but almost halves the error rate. Also note that Boosting SVM takes significantly more time in nursery than a single SVM model because the changed data distribution increases the training time significantly for training individual models. In comparison with Bagging, Feating runs faster on all data sets, without exception.

Table 3 Average error rate for SVM, Boosting SVM, Random Subspace (RS) SVM, Bagging SVM and Feating SVM using $h = 1$. RS, Bagging and Feating have the same ensemble size. Boosting with early stopping (up to 100 iterations) is used here. Entries with boldface (resp. italic) indicate that the error rates are significantly lower (resp. larger) than those of a single SVM, in a pair-wise t-test with 5% level of significance

	Error rate				
	SVM	Boosting	RS	Bagging	Feating
coding	28.58	28.58	*29.67*	28.66	**26.10**
nursery	6.85	6.85	*25.24*	6.91	**3.75**
dna	7.34	7.41	**3.58**	**5.93**	**3.77**
wave40	13.72	13.72	13.52	13.50	13.68
satimage	13.33	13.33	*14.11*	13.35	**11.84**
wave21	12.98	12.98	13.34	13.00	13.00
segment	6.97	6.67	*8.44*	6.93	**6.41**
anneal	12.58	**8.02**	*20.71*	11.92	**9.91**

Table 4 Total execution time (in seconds) for SVM, Boosting SVM, RS SVM, Bagging SVM and Feating SVM using $h = 1$. The average number of boosting iterations over the 10-fold cross validation is also provided. RS, Bagging and Feating have the same ensemble size, shown in the last column

	Total Time					#Iterations	
	SVM	Boosting	RS	Bagging	Feating	Boosting	others
coding	2392	9798	6234	219375	5962	3.9	15
nursery	47	1571	208	2001	59	12.5	8
dna	17	2295	672	2851	251	100.0	60
wave40	3.6	28	23	288	50	3.6	40
satimage	3.5	14	25	236	59	3.1	36
wave21	2.4	14	8	61	15	4.2	21
segment	2.0	9.2	11	62	13	6.5	19
anneal	1.9	18	12	97	20	19.8	38

RS runs faster than Bagging in all data sets and faster than Boosting in six data set; but it is still significantly slower than Feating in the three most time consuming data sets (coding, nursery and dna).

RS is shown to have worked with SVM by Tao et al. (2006) to improve the retrieval performance for relevant feedback in an image retrieval application. Each SVM in RS is trained from a small training set obtained from relevance feedback; the ensemble is shown to perform better than a single SVM trained from a small training set. As far as we know there is no other evidence that RS works well with SVM in the general setting. In particular, there is no evidence that it works well with a large data set where the variance will be low.

Kim et al. (2002) show that Bagging SVM improves the predictive accuracy of SVM with a small margin in two data sets. This result is consistent with our result shown in Table 3 which reports minor improvements in three data sets but no improvements in the other five data sets.

3.3 Feating J48 and Feating IBk

To demonstrate that Feating performs well on both unstable and stable base learners, we present the results of Feating J48 and Feating IBk in Tables 5 and 6 respectively. J48 is a C4.5 decision tree classifier (Quinlan 1993) and is a good representative from unstable base learners, whereas IBk is a k-nearest neighbour algorithm and is a good example of another stable base learner.

Table 5 presents the average error rate results of the experiment using J48 and three levels of localisation for Feating J48. They show that the average error rates decrease monotonically from 13.60% for J48 to 7.95% for Feating J48 at $h = 3$, and similarly the geometric means drop monotonically from .70 (at $h = 1$) to .56 (at $h = 3$). The same trend happens in most of the data sets. The only exception is the segment data set, in which its error rate increases slightly from $h = 2$ to $h = 3$. This small fluctuation is mainly due to the small data size.

Table 5 also summarises the execution time to construct and test a single model. It shows that Feating reduces the execution time for constructing and testing a single model as the

Table 5 Average error rate and execution time (in seconds) for constructing and testing one model of J48 and Feating J48 at $h = 1, 2$ and 3 levels of localisation. Entries with boldface indicate that the error rates are significantly lower than those of a single J48, in a pair-wise t-test with 5% level of significance

	Error rate				Time per model			
	J48	Feating J48			J48	Feating J48		
		$h = 1$	$h = 2$	$h = 3$		$h = 1$	$h = 2$	$h = 3$
coding	27.22	**18.31**	**15.95**	**15.34**	1.27	.51	.34	.30
nursery	1.44	**1.23**	**.89**	**.72**	.55	.23	.15	.13
dna	8.19	**4.61**	**4.11**	**3.26**	.60	.15	.11	.13
wave40	24.90	**18.52**	**16.24**	**15.68**	5.21	2.12	1.86	1.73
satimage	14.30	**8.81**	**7.94**	**7.69**	5.34	1.61	1.22	1.07
wave21	24.08	**17.10**	**15.74**	**15.32**	2.39	1.19	.96	.84
segment	2.99	**2.12**	**1.86**	**2.03**	.87	.50	.23	.16
anneal	5.68	**4.57**	**3.90**	**3.57**	.39	.15	.06	.04
Average	13.60	9.41	8.33	7.95	2.08	.81	.62	.55
Geomean		.70	.61	.56		.39	.25	.22

Table 6 Average error rate and execution time (in seconds) for constructing and testing one model of IBk and Feating IBk at $h = 1, 2$ and 3 levels of localisation. Entries with boldface (resp. italic) indicate that the error rates are significantly lower (resp. larger) than those of a single IBk, in a pair-wise t-test with 5% level of significance

	Error rate				Time per model			
	IBk	Feating IBk			IBk	Feating IBk		
		$h = 1$	$h = 2$	$h = 3$		$h = 1$	$h = 2$	$h = 3$
coding	25.85	**24.25**	**23.01**	**21.82**	22.56	6.00	1.73	.59
nursery	1.72	*2.00*	*2.00*	*2.00*	5.81	1.67	.61	.26
dna	20.09	**12.52**	**8.25**	**4.83**	3.49	.75	.21	.11
wave40	20.16	**18.64**	**17.38**	**16.66**	5.27	3.28	2.36	1.70
satimage	9.11	9.04	8.94	8.89	3.49	2.29	1.43	.98
wave21	17.88	**16.38**	**15.80**	**15.54**	3.24	1.33	.77	.53
segment	4.72	**4.07**	**3.38**	**3.03**	.89	.29	.14	.09
anneal	6.91	6.91	6.79	6.57	.88	.17	.10	.08
Average	13.30	11.73	10.69	9.92	5.70	1.97	.92	.54
Geomean		.92	.83	.75		.34	.16	.09

level of localisation increases. The geometric mean ratios indicate that Feating at $h = 1$ only needs 39% of the time to construct and test a model of J48 and this can be further reduced to 22% at $h = 3$. The reduction in execution time over a higher level of localisation occurs monotonically in most data sets.

The same trends are also observed from the results of IBk and Feating IBk in Table 6—the error rate and execution time also decrease with increasing levels of localisation. In fact, the amount of execution time reduction achieved by Feating IBk (at $h = 3$) is reduced to just a mere 9% of IBk.

The above results further show that Feating is a general ensemble method that works well for both stable and unstable learners.

3.4 Why use Feating instead of Boosting or Random Forests?

Our results in the previous three sections show that Feating has two key advantages relative to other ensemble approaches: (i) the reduction in execution time per model as the level of localisation increases; and (ii) it works well with both stable and unstable learners. Therefore, Feating is useful when the data size is large or the computational complexity of the base learner is superlinear with respect to the number of training examples, and it is particularly useful in domains where stable learners outperform the alternatives. Table 7 shows three such example domains where SVM with quadratic kernels (SVMq) is a more accurate base learner than J48 and SVM (with linear kernels); and Feating (with randomisation) can be used to further improve its predictive performance. The result in Fig. 3 shows that Feating SVMq performs significantly better than Boosting J48 and Random Forests, two commonly used ensembles, in all three data sets.

Boosting SVMq has an error rate of 21.72% in the usps data set, there is no improvement to the error rate of the single SVMq (which also has 21.72% error rate). Furthermore, Boosting SVMq on the pendigits data gives a higher error rate of 2.72% than 2.1% for a single SVMq. These poor results are not surprising because boosting is known to work for unstable models only (Breiman 1998). Previous approaches that have applied boosting to

Table 7 The three data sets in which SVM is more accurate among several base learners used in this study. The randomised version of Feating must be used, especially in the first two data sets where $|Rh|$ is prohibitively huge. All experiments for these data sets use one run with the given training and testing sets

Data set	#Instances train/test	#Classes	Feating's Ensemble size		
			$h=1$	$h=3$	$h=n/2$
connect-4	50000/17557	3	42	11480	5.38×10^{11}
usps	7291/2007	10	256	2763520	5.77×10^{75}
pendigits	7494/3498	10	16	560	12870

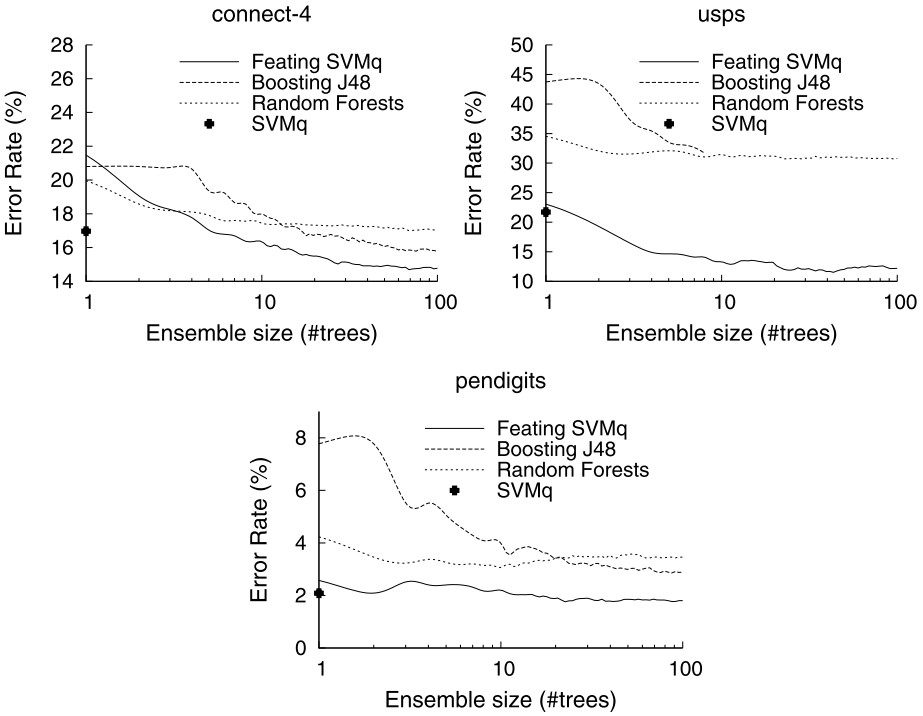

Fig. 3 Error Rate Curves for Feating SVMq, Boosting J48 and Random Forests. SVMq uses quadratic kernels. Feating SVMq reduces SVMq's error rate from 2.1% to 1.8% in pendigits; Boosting stops early at 8 iterations in usps. Randomised Feating is used here

SVM have used it to select a subsample to scale up the efficiency of SVM for large data (Pavlov et al. 2000) or improve predictive accuracy for small data sets only (Li et al. 2005). The boosting procedure is restarted when the termination condition is met (Li et al. 2005), and different values of the kernel parameter are used in order to increase model diversity—this increases the training time of a normal boosting process without reducing the training time for individual SVM, which is infeasible for large data sets.

In contrast, Feating not only scales up the efficiency of SVM, but also significantly improves its predictive performance, as we have shown in Sect. 3.1. For the most time consuming connect-4 data set shown in Table 8, Feating SVMq of 100 models takes 1.7 days—which is about one-tenth of the time (16.3 days) to train one single SVMq model! Boosting

Table 8 Total execution time (in seconds) for Single SVMq, Feating SVMq, Random Forests, Boosting J48 and Boosting SVMq for an ensemble size of 100 models, unless otherwise stated. The time for Boosting SVMq on connect-4 data set is not available (N.A.) because its runtime is extremely long. The number of iterations carried out by Boosting are in square brackets. Randomised Feating is used here

Data set	SVMq	Feating SVMq	Random Forests	Boosting	
				J48	SVMq
connect-4	1413248	151461	6660	504 [100]	N.A.
usps	263	2614	49213	1717 [8]	829 [1]
pendigits	17	633	338	118 [100]	3779 [40]

SVMq of 100 models in this data set is estimated to take about 4.5 years! Boosting has no mechanism to reduce the runtime. Boosting or other ensemble methods that build global models would be infeasible for algorithms with high computational time such as SVMq in large data sets. Note that while it is possible to reduce the runtime by tuning the machine characteristic to speed up the runtime of a specific base learner, the key difference of ensemble methods between Feating and Boosting in terms of their ability to reduce runtime is independent of this machine tuning.

Note that despite being a tree-based ensemble, Random Forests still take about half of the time required by Feating SVMq in the pendigits data set. In the connect-4 data set which has a high number of instances, Feating SVMq takes about 20 times longer than Random Forests; but in the usps data set with a high number of features, Random Forests take close to 20 times longer than Feating SVMq.

3.5 Feating (with randomisation) using SVMq and J48

To demonstrate the flexibility of Feating, we apply the randomised version of Feating to use SVMq and J48 base learners on the eight data sets. Figure 4 shows that Feating SVMq and Feating J48 produce a lower error rate than their respective base learners; the only two exceptions are wave21 and nursery, where Feating SVMq's error rates are marginally higher than that of SVMq.

These results also reveal the need to use different types of base learners in different domains. Feating J48 outperforms Feating SVMq in half of the data sets, i.e., segment, anneal, satimage and coding. For the other four remaining data sets, Feating SVMq outperforms Feating J48. To improve predictive accuracy, we can readily switch the base learner from J48 to SVMq or vice-versa. There is no need to change to another ensemble method in order to use a different type of base learner, or to introduce an additional mechanism to increase the instability of SVM. This feature of Feating is a significant advantage over many existing ensemble methods, which are mainly restricted to using unstable models.

4 Discussion

4.1 Algorithmic issues in Feating

Here we will discuss five algorithmic issues that affect the operation of Feating: h and n_{min} settings, missing value treatment, aggregation method, and attribute ranking.

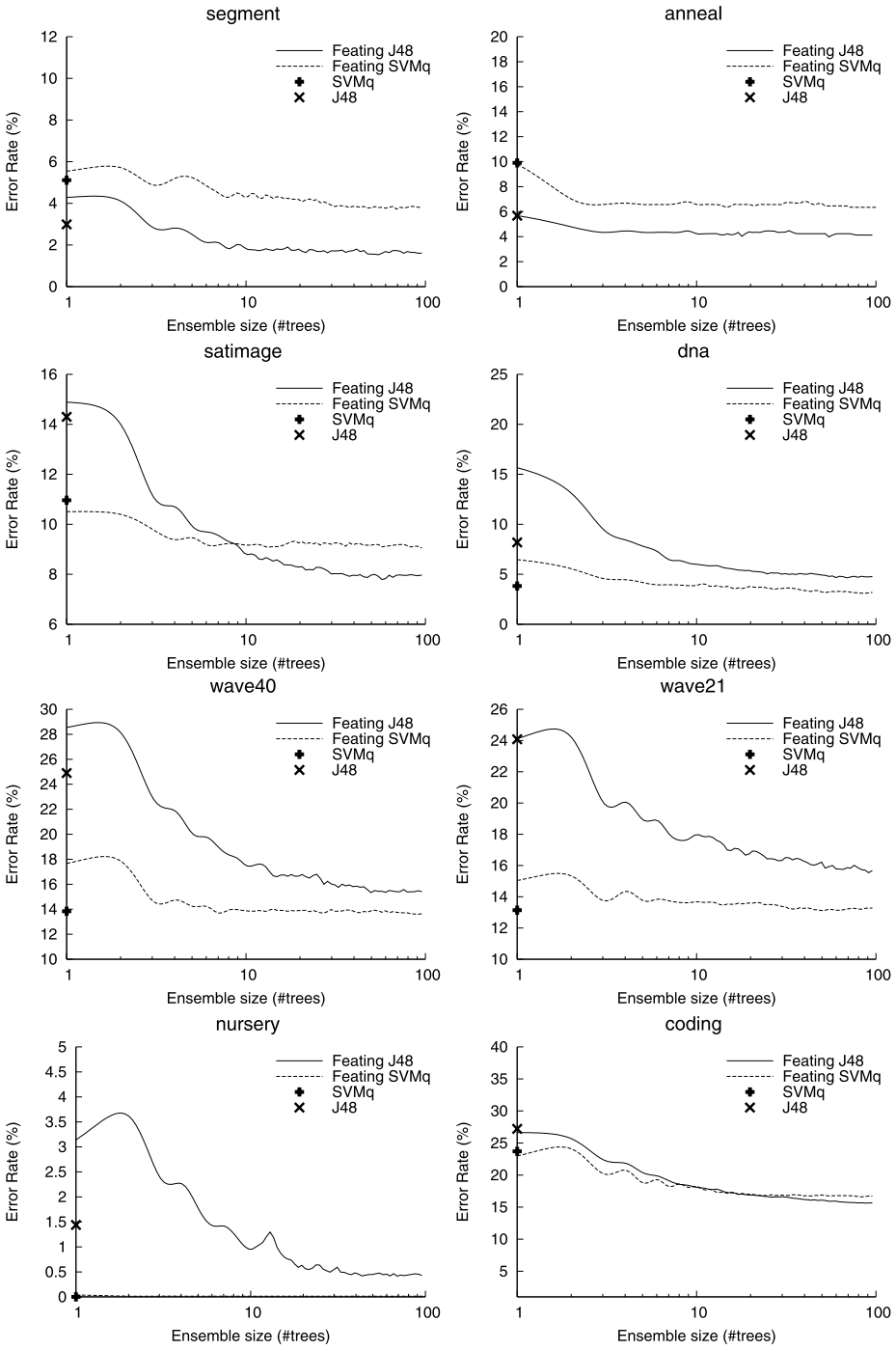

Fig. 4 Error Rate Curves for Feating J48 and Feating SVMq. The error rate for Feating SVMq in the nursery data set is very small (about 0.015%), so its error rate curve is not visible from the plot. Randomised Feating is used here. The result for each ensemble size in each curve is obtained from one run of train and test, where 10% of data is randomly selected as a test set and the remaining 90% is used as a training set

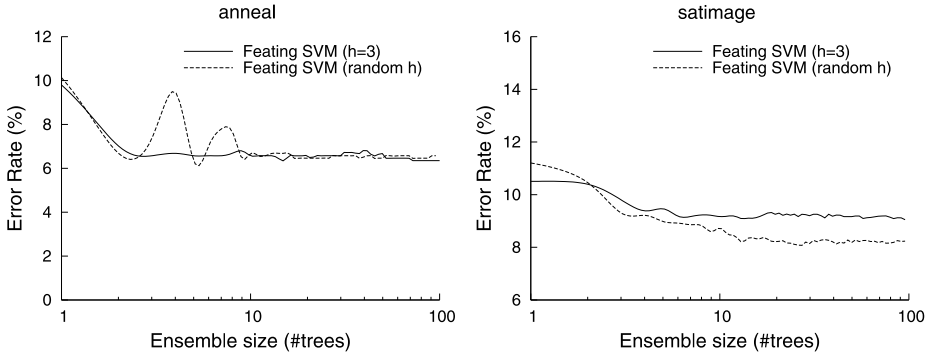

Fig. 5 Error Rate Curves for Feating SVM: $h = 3$ versus random h. Randomised Feating is used and the result for each ensemble size in each curve is obtained from one run of train and test with a 9:1 random split of the given data set

The setting of h in a particular data set is influenced by four factors: (i) training data size m, (ii) branching factor b in Level Tree, (iii) the base learner employed, and (iv) computing resources in terms of time and space. Because a high level of localisation is preferred, we first discuss how to determine the upper limit and then suggest ways to set a lower value if necessary.

We have set the upper limit of $h = n/2$ in Sect. 2.1 because it produces the largest ensemble size C_h^n. While it is possible to use a higher h value, this has two disadvantages: the tree becomes too large and the data size at each leaf may be too small to train an accurate model.

A rule of thumb to set the upper limit (lower than $h = n/2$ for large n) is $h < \log_b \frac{m}{M}$, assuming b-nary Level Trees are built from a training set of size m, where M is the minimum data size required in order to build an accurate local model. This explains the poor predictive performance for Feating SVM in the dna data set when $h = 3$ (shown in Table 2). If we set $M = 100$ for SVM, then the upper limit should be $h < \log_4 \frac{3186}{100} = 2.5$. As a result of setting $h = 3$, many local models are trained from small data sizes and consequently have low accuracy.

Here we provide two ways in which h may be set in practice, if the computing resources do not allow the upper limit suggested above to be used. First, especially for the randomised version of Feating, it is possible to avoid setting an h value altogether by using an h value selected at random from $[2, n/2]$ to build each model in the ensemble, resulting in a mixture of models built from a range of h values in the ensemble. Figure 5 shows two examples in which Feating SVM using random h performs equally well or better than that using $h = 3$. It shall be cautioned that while this works for many data sets we have used, it does not work for all data sets.

Second, one may employ a binary search to find an appropriate h value from $[1, n/2]$ for a particular data set—start with $h = n/2$ or a reasonable value if $n/2$ is too large; and end the search when the predictive accuracy begins to decline. We find that an ensemble of 10 models is sufficient to provide an indication of its final performance for each h in the search process. For example, in the three data sets described in Sect. 3.4, Feating SVMq uses $h = 21$, $h = 50$ and $h = 3$ in connect-4, usps and pendigits, respectively, after a binary search. In the case of usps, we start with $h = 50$ (instead of $n/2 = 128$) because $n/2$ is too large; it is possible that a higher h could produce a better result in this data set. The highest h values are found in just two examinations in the binary search for connect-4 and usps.

The default setting of n_{min} is 4 so that as many as possible of the branches of the Level Tree are grown to the user specified height. If we were to relax the Rh condition, then n_{min} may be set equal to M, the minimum data size required in order to build an accurate local model. Alternatively, we could build a local model at a leaf at level h only if it has at least M training instances; otherwise a majority vote is employed instead. These variants may improve the predictive performance and reduce the runtime of Feating reported in this paper.

Missing values in a test instance are treated as follows. If there is no branch for missing values (a result of no missing values in the training set), then the majority class at the node is used as the predicted class. This treatment is the same when there is a leaf with no training data, the majority class of the parent node is used as the predicted class. In the only data set which has missing values, i.e., anneal (where twenty out of thirty-eight attributes have more than 90% values missing), this simple treatment has served pretty well in Feating, as evidenced from its results in this data set. However, there may be room for improvement. For example, instead of predicting using majority vote, one may opt to aggregate predictions made by local models only and ignore predictions which cannot be made using local models. DePasquale and Polikar (2007) have shown promising results using this selective aggregation in ensembles.

Feating uses a simple majority vote when aggregating results from individual models. A weighted majority vote approach (Cerquides and Mantaras 2005; Yang et al. 2007) can potentially be beneficial too.

The attribute ranking in Algorithm 1 is purely for the purpose of enumeration in the way we have described in Sect. 2.1. The ranking is not required if the randomised version of Feating is used or Feating always uses all C_h^n models. However, if a subset of Rh is used (in the enumerated version of Feating) which comes from the initial subset of the enumeration, then, the order is important. It shall be noted that using the reverse order to that provided by information gain produces a poor Feating result in our current implementation. This is because attributes with high information gains produce more homogeneous subregions, thus better performing local models in the initial subset of the enumeration.

4.2 Other issues

Irrelevant attributes have no impact on Feating's ability to reduce predictive error rate through increased localisation. This is demonstrated in the wave40 data set in Tables 2, 5 and 6. This data set is the version of the wave21 data set with 19 additional irrelevant attributes. Feating SVM, J48 and IBk in wave40 achieves a similar level of predictive performance as that in wave21, as the level of localisation increases to $h = 3$. In fact, irrelevant attributes employed in Level Trees serve to reduce their effect such that local models are built without these irrelevant attributes. In other words, given sufficient data at the local regions, local models built under Level Trees can be expected to perform better than a global model which needs to battle with more irrelevant attributes.

Our work on the ensemble of stable learners also complements recent SVM research. For example, Tsang et al. (2005, 2007) and Fan et al. (2008) have shown that it is possible to improve the runtime of SVM significantly; and Yu et al. (2010) have devised a SVM method when data cannot fit into memory. Despite these advances, SVM are still stable learners which prohibit them from being a successful base learner in an ensemble, when using existing ensemble methods. Feating remains the only ensemble method that can improve the predictive accuracy of SVM and its execution time per model.

There has been little previous research into ensembles for stable learners. The bootstrap model average method introduced by Davidson (2004) creates global models from multiple

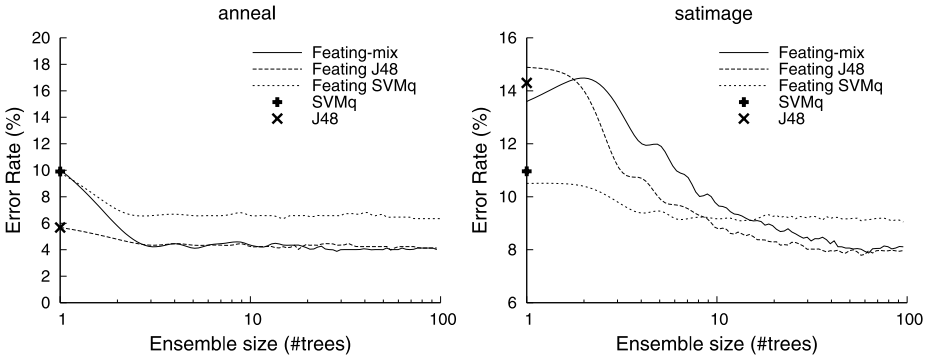

Fig. 6 Error Rate Curves for Feating-*mix*, Feating J48, and Feating SVMq, where $h = 3$ is used in all cases. Randomised Feating is used and the result for each ensemble size in each curve is obtained from one run of train and test with a 9:1 random split of the given data set

bootstrap samples to form an ensemble of Naive Bayes. Because bootstrap samples cannot create sufficient model diversity in stable learner, the improvement (mainly due to model averaging) is small in a practical sense, even though the results can be statistically significant. Similar results when applied to bagging SVM are shown in Kim et al. (2002). In another work (Tao et al. 2006), bagging is used in conjunction with random subspacing to increase SVM's instability with the aim to improve predictive accuracy. The use of global models in these approaches means that the methods will struggle to work for large data sets because, unlike Feating, they cannot reduce the execution time required to train individual models.

A number of researchers have investigated local learning in a single global model only (e.g., Atkeson et al. 1997; Klanke et al. 2008). They perform local weighting to avoid unnecessary bias of a single global function. To the best of our knowledge, Feating is the first approach to ensemble local models.

Previous approaches to increasing model diversity mainly focus on manipulating the instance space (e.g., bagging and boosting), randomising the model building process (e.g., Random Forests) or feature subspacing at the global level (e.g., Random Subspace.) In contrast, model diversity in Feating is derived from different subsets of instances and different feature subsets at the local level most appropriate to every test instance. In addition to the ability to employ either stable or unstable base learners, Feating opens a new way to increasing model diversity—by allowing a mixture of stable and unstable base learners in a single ensemble. For example, Feating may be used to include both SVM and J48—all within a single ensemble. This can be implemented by randomly selecting either SVM or J48 with equal probability under the current randomised version of Feating. In a situation in which extensive experimentations cannot be afforded, this implementation could perform close to the better performing Feating SVM or Feating J48. Figure 6 shows two such examples in which Feating SVM or Feating J48 performs better than the other in different data sets, yet Feating-*mix* (i.e., Feating mixing SVM and J48) delivers a performance similar to the better one in both cases, without the need to select a specific base learner. Furthermore, Feating-*mix* requires a runtime in-between those taken by Feating J48 and Feating SVM. For example, Feating-*mix* takes 27 seconds, whereas Feating J48 and Feating SVM take 12 and 127 seconds respectively in the anneal data set.

Relaxation of the attribute independent assumption of Naive Bayes has been the initial motivation for AODE (Webb et al. 2005) where the base learner is a Naive Bayes classifier, with the aim to reduce attribute independence assumption. Since Feating is a generalisation

of AODE and it is applicable to learners with or without an attribute independence assumption, this suggests that attribute inter-dependency is not critical to the success of the Feating approach. Indeed, the success of Feating with added random variables (wave40) provides further support for this conclusion. Rather, a complete enumeration of all possible local regions for every point in the feature space ensures successful ensembling because (i) it provides a maximum diversity of models at every point in the feature space; and (ii) it includes local models in the ensemble that take account (albeit implicitly) of each combination of features in turn (those in the Level Tree branch leading to the local model), irrespective of the base learner's learning biases.

5 Future work

This work opens up many avenues for future research. First, the 'distinct local regions'—*Rh* condition can be taken in two opposite directions: (i) Relaxation of the condition—it is unclear how far we can relax the condition before the Feating approach breaks down. The randomised version of Feating has a small degree of relaxation. It would be interesting to see whether a complete relaxation will work—by using a completely random tree (Liu et al. 2008) and then build local models at the leaves. (ii) Maintaining the condition—a possible way to uphold the condition while further increasing model diversity under Feating is to aggregate all local models produced through different projections, in addition to those provided in the original feature space.

Second, instead of defining a subspace using a rule, one may use a functional to subdivide the feature space. The functional employed shall possess a parameter akin to h which enables a user to adjust the desired level of localisation.

Third, it may be worthwhile to search for a subset of local regions. The search may pay off with a smaller ensemble that performs better than the one using the whole set or a random subset. This can be achieved using either a pre-pruning method or a post-pruning method.

Fourth, Level Trees are built solely based on the characteristics of the feature space (i.e., feature types and available values), independent of the complexity of the problem or base learner used. There may be merit in influencing how the local regions are constructed (dependent on the base learner used) in order to build a better performing ensemble, albeit at a computational cost. For example, subsumption resolution (Zheng and Webb 2006) has demonstrated the potential value of exploiting attribute inter-dependencies within AODE, where Naive Bayes is employed. It would be interesting to explore whether such techniques are applicable within the more general Feating approach. This issue is closely related to the third issue we have discussed above, which could possibly be tackled in one step.

Fifth, Feating is designed for large data sets. Small data size does not allow high h to be used; this limits the model diversity and algorithmic speedup that can be achieved otherwise. It may be possible to use some feature reduction method (e.g., random projection (Bingham and Mannila 2001)) so that the number of features is reduced to a size commensurate with the data size for Feating to work in small data sets.

Sixth, our findings with regard to random h setting and Feating-*mix* (reported in Sects. 4.1 and 4.2, respectively) need further investigation to ascertain the conditions under which they will perform well.

6 Conclusions

This paper shows the advantages of using local models in ensembles. This is a significant departure from existing ensemble methods that utilise global models through randomisa-

tion (or perturbation) on the training data or of the training process. The proposed Feating approach possesses the same property of existing ensemble approaches, i.e., increasing ensemble size improves an ensemble's predictive performance, with two added advantages.

First, Feating significantly reduces the execution time for each model as the level of localisation increases. Our analysis shows that the ratio of reduction is in the order of $b^{h(p-1)}$ when an algorithm with time complexity $O(m^p)$ is employed to generate local models in Feating's b-nary Level Trees of h-subdivision. In other words, the reduction becomes more pronounced when the algorithm has high order polynomial time complexity with respect to the size of the training set.

Our empirical results show that Feating learns each model faster—Feating is an effective way to speed up SVM to make ensembles feasible for SVM; and Feating SVM is significantly faster than Boosting SVM, Random Subspace SVM and Bagging SVM for large data sets. We also show another unique feature of Feating that is not available to other ensemble methods: building multiple SVM models can both reduce learning time and improves accuracy relative to a single SVM model.

Second, Feating is an ensemble approach which is more generic than existing approaches; many of which are limited to unstable models. Both stable and unstable models such as SVM and decision trees can be used in Feating to improve their predictive performance by increasing the ensemble size through increasing level of localisation. This advantage enables a mixture of stable and unstable base learners to easily form an ensemble—Feating-*mix*, which contains both J48 and SVM local models, achieves the better predictive accuracy of Feating J48 and Feating SVM. In addition, we demonstrate the utility of Feating over existing strong ensembles—Feating SVMq outperforms Boosting J48 and Random Forests when SVMq is the preferred base learner, in comparison to a decision tree.

These advantages are a direct result of using local models rather than global models in the ensembles, and the Feating approach ensures that there are no duplicate local regions in which the same data is used to generate multiple local models in the ensemble—ensuring maximum model diversity.

Acknowledgements We would like to thank the anonymous reviewers who had provided many helpful comments to improve this paper.

References

Asuncion, A., & Newman, D. J. (2007). *UCI repository of machine learning databases*. University of California, Irvine, CA.

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review*, *11*, 11–73.

Bingham, E., & Mannila, H. (2001). Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 245–250). New York: ACM.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*, 123–140.

Breiman, L. (1998). Arcing classifiers (with discussion). *Annals of Statistics*, *26*(3), 801–849.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32

Cerquides, J., & Mantaras, R. L. D. (2005). Robust Bayesian linear classifier ensembles. In *Proceedings of the sixteenth European conference on machine learning* (pp. 70–81). Berlin: Springer.

Davidson, I. (2004). An ensemble technique for stable learners with performance bounds. In *Proceedings of the thirteenth national conference on artificial intelligence* (pp. 330–335). Menlo Park: AAAI Press.

DePasquale, J., & Polikar, O. (2007). Random feature subset selection for ensemble based classification of data with missing features. In *Lecture notes in computer science: Vol. 4472. Multiple classifier systems* (pp. 251–260). Berlin: Springer.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin, C.-J. (2008). LIBLINEAR: a library for large linear classification. *Journal of Machine Learning Research*, *9*, 1871–1874.

Frank, E., Hall, M., & Pfahringer, B. (2003). Locally weighted Naive Bayes. In *Proceedings of the 19th conference on uncertainty in artificial intelligence* (pp. 249–256). San Mateo: Morgan Kaufmann.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Transactions Pattern Analysis and Machine Intelligence*, *20*(8), 832–844.

Kim, H.-C., Pang, S., Je, H.-M., Kim, D., & Bang, S.-Y. (2002). Support vector machine ensemble with bagging. In *Lecture notes in computer science: Vol. 2388. Pattern recognition with support vector machines* (pp. 131–141). Berlin: Springer.

Klanke, S., Vijayakumar, S., & Schaal, S. (2008). A library for local weighted projection regression. *Journal of Machine Learning Research*, *9*, 623–626.

Kohavi, R. (1996). Scaling up the accuracy of Naive-Bayes classifiers: a decision-tree hybrid. In *Proceedings of the 2nd international conference on knowledge discovery and data mining* (pp. 202–207). New York: ACM.

Kohavi, R., & Li, C. H. (1995). Oblivious decision trees, graphs, and top-down pruning. In *Proceedings of the 14th international joint conference on artificial intelligence* (pp. 1071–1077). San Mateo: Morgan Kaufmann.

Li, X., Wang, L., & Sung, E. (2005). A study of AdaBoost with SVM based weak learners. In *Proceedings of the international joint conference on neural networks* (pp. 196–201). New York: IEEE Press.

Liu, F. T., Ting, K. M., Yu, Y., & Zhou, Z. H. (2008). Spectrum of variable-random trees. *Journal of Artificial Intelligence Research*, *32*, 355–384.

Opitz, D. (1999). Feature selection for ensembles. In *Proceedings of the 16th national conference on artificial intelligence* (pp. 379–384). Menlo Park: AAAI Press.

Oza, N. C., & Tumer, K. (2001). Input decimation ensembles: decorrelation through dimensionality reduction. In *LNCS: Vol. 2096. Proceedings of the second international workshop on multiple classifier systems* (pp. 238–247). Berlin: Springer.

Pavlov, D., Mao, J., & Dom, B. (2000). Scaling-up support vector machines using the boosting algorithm. In *Proceedings of the 15th international conference on pattern recognition* (pp. 219–222). Los Alamitos: IEEE Comput. Soc.

Quinlan, J. R. (1993). *C4.5: program for machine learning*. San Mateo: Morgan Kaufmann.

Schapire, R. E., & Singer, S. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, *37*, 297–336.

Tao, D., Tang, X., Li, X., & Wu, X. (2006). Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(7), 1088–1099.

Tsang, I. W., Kwok, J. T., & Cheung, P.-M. (2005). Core vector machines: fast SVM training on very large data sets. *Journal of Machine Learning Research*, *6*, 363–392.

Tsang, I. W., Kocsor, A., & Kwok, J. T. (2007). Simpler core vector machines with enclosing balls. In *Proceedings of the twenty-fourth international conference on machine learning* (pp. 911–918). San Mateo: Morgan Kaufmann.

Webb, G. I., Boughton, J., & Wang, Z. (2005). Not so naive Bayes: averaged one-dependence estimators. *Machine Learning*, *58*(1), 5–24.

Witten, I. H., & Frank, E. (2005). *Data mining: practical machine learning tools and techniques* (2nd edn.). San Mateo: Morgan Kaufmann.

Yang, Y., Webb, G. I., Cerquides, J., Korb, K., Boughton, J., & Ting, K. M. (2007). To select or to weigh: a comparative study of linear combination schemes for superparent-one-dependence ensembles. *IEEE Transaction on Knowledge and Data Engineering*, *19*(12), 1652–1665.

Yu, H.-F., Hsieh, C.-J., Chang, K.-W., & Lin, C.-J. (2010). Large linear classification when data cannot fit in memory. In *Proceedings of the sixteenth ACM SIGKDD conference on knowledge discovery and data mining* (pp. 833–842). New York: ACM.

Zheng, Z., & Webb, G. I. (2000). Lazy learning of Bayesian rules. *Machine Learning*, *41*(1), 53–84.

Zheng, F., & Webb, G. I. (2006). Efficient lazy elimination for averaged one-dependence estimators. In *Proceedings of the twenty-third international conference on machine learning* (pp. 1113–1120). San Mateo: Morgan Kaufmann.