

**Title:**

Recursive-Stacking To Improve The Accuracy of Combined Classifiers

**Author(s) with address(es):**

David W. Fan  
Salvatore J. Stolfo  
Department of Computer Science  
Columbia University  
New York, NY 10027

Philip K. Chan  
Computer Science  
Florida Institute of Technology  
Melbourne, FL 32901

**Abstract (250 word maximum):**

We analyze the mechanism of stacking and point out the conflict problem. Two methods to reduce conflicts are discussed and their equivalence is established. We propose the Recursive-Stacking Algorithm to solve conflicts and demonstrate its effectiveness using two artificial data sets and two real world data sets.

**Keywords:** Conflict, Stacking, Recursive Stacking

**Email address of contact author:** wfan@cs.columbia.edu

**Phone number of contact author:** 212-939-7078

**Multiple submission statement (if applicable):** N/A

# 1 Introduction

*Integrating multiple learned models* has been identified as one of the current research directions in machine learning [7]. Empirical studies in the last 5 years have shown that integrating multiple learned models helps to increase both accuracy and efficiency. There are different techniques, but within the scope of combining classifiers only [7], there are mainly three proposed methods: voting (unweighted and weighted) [5, 11], gating function [12] and stacking [15]. The goal of stacking is to learn a combining classifier that reflects the correlation of predictions of underlying base classifiers that are combined.

Following Wolpert[15], the general scheme for stacking works as follows. We have  $t$  different algorithms  $A_1, \dots, A_t$  and a set  $\mathcal{S}$  of training examples  $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$ .  $\mathcal{S}$  is CV partitioned into  $n$  pairs  $(\mathcal{T}_1, \mathcal{V}_1), \dots, (\mathcal{T}_n, \mathcal{V}_n)$  ( $2 \leq n \leq |\mathcal{S}|$ ).  $\mathcal{T}_k$  is called the stacking training set and  $\mathcal{V}_k$  is the validation set. We then apply each of the learning algorithms on  $\mathcal{T}_k$  to form classifiers  $C_1$  to  $C_t$  and apply them on  $\mathcal{V}_k$  to obtain predicted class labels.  $\forall (\mathbf{x}_i, y_i) \in \mathcal{V}_k$ , we form a new training item :  $((C_1(\mathbf{x}_i), C_2(\mathbf{x}_i), \dots, C_t(\mathbf{x}_i)), y_i)$ . The process is repeated for all  $n$  pairs of  $(\mathcal{T}_k, \mathcal{V}_k)$  to form a new data set,  $\mathcal{M}_{train}$ . We then train  $A_1$  to  $A_t$  on  $\mathcal{S}$  to produce base classifiers  $C_1$  to  $C_t$ . We may use any algorithm to learn the new *meta-level* training set,  $\mathcal{M}_{train}$ , to generate the meta-classifier  $M_C$ . During testing, for a data item with unknown label  $(\mathbf{x}, ?)$ ,  $C_1 \dots C_t$  first generate a prediction. Their predictions form a meta-level testing item  $(C_1(\mathbf{x}), \dots, C_t(\mathbf{x}))$  that is fed into the meta-classifier. The meta-classifier's prediction  $M_C((C_1(\mathbf{x}), \dots, C_t(\mathbf{x})))$  is the final outcome.

The effectiveness of stacking to increase accuracy has been widely reported [4, 14, 15]. Stacking is also a feasible approach to learn over inherently distributed data [4]. The idea of stacking has been deployed and implemented as agents to attack credit card fraud [13]. Experiments in different domains have shown accuracy increases from 0.5% to 3% over any of the combined base classifiers. In this paper, we will discuss a problem called *conflicts* that hinders the accuracy improvement of stacking. We propose and discuss some methods to solve it. Conflicts were first discussed in [9, 14].

In Section 2, we analyze the mechanism of stacking and show the conflict problem. We will discuss two seemingly different approaches, and their intrinsic equivalence. The Recursive-Stacking Algorithm to reduce conflicts is proposed. In Section 3, we analyze experimental results. Section 4 will discuss the effectiveness and efficiency of Recursive-Stacking.

## 2 Problem and Approaches

### 2.1 Conflict

For simplicity, we consider bimodal problems (2 classifications, 0 and 1). The prediction  $C_i(\mathbf{x})$  of classifier  $C_i$  on data item  $(\mathbf{x}, y)$  is abbreviated in lowercase  $c_i$ .  $((c_1, c_2, \dots, c_t), y)$  is thus a meta-level training data item (level 1 training data in the terminology of Wolpert[15]). The feature  $(c_1, c_2, \dots, c_t)$  is a vector of predictions generated by base learning algorithms on an unseen item  $(\mathbf{x}, y)$ , and  $y$  is the true label (either 1 or 0). Each training data item  $(\mathbf{x}, y)$  in  $\mathcal{S}$  will fall into exactly one combination of  $(c_1, c_2, \dots, c_t)$ , since each is classified exactly once in stacking’s CV partition. For bimodal problems, there are at most  $2^t$  combinations. Figure 1 shows the 8 combinations of 2 boolean data sets (see Section 3). To understand how stacking works, for a fixed combination  $(c_1, c_2, \dots, c_t)$ , we count all the occurrences of the same instance  $((c_1, c_2, \dots, c_t), 1)$  in the meta-level training data (denoted  $|((c_1, c_2, \dots, c_t), 1)|$ ), and all occurrences of  $((c_1, c_2, \dots, c_t), 0)$  (denoted  $|((c_1, c_2, \dots, c_t), 0)|$ ). We define *accuracy*  $\alpha$  and *conflict ratio*  $\epsilon$  on combination  $(c_1, c_2, \dots, c_t)$  as:

$$\epsilon = \frac{\min(|((c_1, c_2, \dots, c_t), 1)|, |((c_1, c_2, \dots, c_t), 0)|)}{|((c_1, c_2, \dots, c_t), 1)| + |((c_1, c_2, \dots, c_t), 0)|}$$

$$\alpha = \frac{\max(|((c_1, c_2, \dots, c_t), 1)|, |((c_1, c_2, \dots, c_t), 0)|)}{|((c_1, c_2, \dots, c_t), 1)| + |((c_1, c_2, \dots, c_t), 0)|}$$

$\epsilon$  and  $\alpha$  have the properties:  $\epsilon + \alpha = 1$ ,  $\alpha \geq \epsilon$ ,  $\alpha \geq 0.5$  and  $\epsilon \leq 0.5$ <sup>1</sup>. For conflict (0, 1, 1) of Boolean.5678 in Figure 1,  $|((ripper, cart, id3), 1)| = 1247$  and  $|((ripper, cart, id3), 0)| = 1445$ , so  $\alpha = 1445/(1247 + 1445) = 0.537$  and  $\epsilon = 1247/(1247 + 1445) = 0.463$ .

Since the feature vector  $(c_1, c_2, \dots, c_t)$  is the same for both  $((c_1, c_2, \dots, c_t), 0)$  and  $((c_1, c_2, \dots, c_t), 1)$ , any machine learning algorithm has no way to distinguish between them. The best it can do is to pick the label of the majority class, and the minority occurrences will always be labelled incorrectly. For the (0,1,1) type discussed above, during testing, whenever the meta-classifier  $\mathcal{M}_C$  sees a vector of (0,1,1), its prediction will always be 0 because  $|((0, 1, 1), 0)|$  is the majority in training the level-1 generalizer.

---

<sup>1</sup>If  $|((c_1, c_2, \dots, c_t), 1)| + |((c_1, c_2, \dots, c_t), 1)| = 0$ , we make  $\alpha = \epsilon = 0$

Boolean.5678 Meta Level Data					
Conflict	$num_1$	$num_0$	$\alpha$	$\epsilon$	Label
(0,0,0)	524	3990	0.884	0.116	0
(0,0,1)	379	639	0.628	0.372	0
(0,1,0)	748	1678	0.692	0.308	0
(0,1,1)	1247	1445	0.537	0.463	0
(1,0,0)	886	284	0.757	0.243	1
(1,0,1)	1318	456	0.743	0.257	1
(1,1,0)	2210	526	0.808	0.192	1
(1,1,1)	11478	1683	0.872	0.128	1

Boolean.45 Meta Level Data					
Conflict	$num_1$	$num_0$	$\alpha$	$\epsilon$	Label
(0,0,0)	867	21929	0.962	0.038	0
(0,0,1)	396	1314	0.768	0.232	0
(0,1,0)	0	0	0.00	0.00	0
(0,1,1)	0	0	0.00	0.00	0
(1,0,0)	1249	1385	0.526	0.474	0
(1,0,1)	1419	932	0.604	0.396	1
(1,1,0)	0	0	0.00	0.00	0
(1,1,1)	0	0	0.00	0.00	0

Note:

$$num_1 = |((ripper, cart, id3), 1)|; num_0 = |((ripper, cart, id3), 0)|.$$

Figure 1: Sample of Conflicts

Stacking gives a final prediction according to the value of  $(c_1, c_2, \dots, c_t)$ . We call each  $(c_1, c_2, \dots, c_t)$  a *type*. A data item is mapped into exactly one type. In each type, we call one classification the majority label if its count is the majority. Stacking chooses the majority label. For data items of type  $(c_1, c_2, \dots, c_t)$ , stacking will have accuracy of  $\alpha$ .

For data of each type  $(c_1, c_2, \dots, c_t)$ , there is always an  $\epsilon$  portion that are labelled wrong. Since the training data and testing data are of the same distribution, this means that the  $\epsilon$  portion will always be misclassified. We call the  $\epsilon$  portion data of each type *conflicts*. It is the  $\epsilon$  portion of conflicts that affects the accuracy increase. Our aim is to look for solutions to reduce  $\epsilon$  in order to increase accuracy of stacking.

To see the difficulty of the problem, consider Figure 1. In the left table, the accuracy  $\alpha$  on (0,0,1) to (1,1,0) are very low: 0.537 to 0.808. There are in total 5647 conflicts out of 29491 training data items. So the maximal accuracy is around 0.8085  $(1.0 - 5647/29491)$  under the assumption that training and testing data are of the same distribution. Another interesting observation is that the stacking algorithm's majority selection is exactly the same prediction by the leftmost classifier (RIPPER), which is the most accurate. ID3 and CART have no contribution to accuracy at all. The second table is even worse, the accuracy of types (0,0,1) to (1,1,0) are from 0.526 to 0.768. The type (1,0,0) is very tricky, since  $\epsilon$  is nearly 0.50, the chance that the prediction will be wrong is high. As we shall see in Section 3, stacking does not even do better than RIPPER. In this example, there are types  $((0,1,0), (0,1,1), (1,1,0)$  and  $(1,1,1))$  having no data items in them. The best stacking can do is to provide the majority label of all the training data, which is 1.

We enumerate all the possible patterns in the meta-level training data. Applying any algorithm on the intrinsically confusing data only, more or less, tries to approach the best we can do, i.e. pick the majority. Algorithms with post-learning pruning or using summarization may remove statistically insignificant conflict types completely, but they do not solve the conflict problem.

Recent publications on stacking [1, 2] have concentrated effort on using metrics (*coverage, diversity, specialty, correlated error*) to choose classifiers for combining and to explain how stacking increases accuracy. Here, we are concerned with the mechanism of stacking to see what contributes to accuracy increase and what prevents it. Conflicts is not a metric. It is the cause of poor performance of stacking. A companion paper [10] discusses the relationship between conflicts and these metrics both conceptually and statistically.

## 2.2 Approaches

We propose two ways to solve the problem of conflicts: *conflict resolver* and *adding base classifiers*. Conflict resolver is a classifying system that is trained on a subset of original training data items that contribute to a particular type of conflict. For each conflict type  $(c_1, c_2, \dots, c_t)$ , we form a subset of training data that generate that conflict by taking all those base level training data items labelled as  $(c_1, c_2, \dots, c_t)$  by base classifiers. We have 8 subsets as displayed in Figure 1. For conflict type (0,1,0), the size of the subset is 2426 (748+1678). All subsets are disjoint and their union is the original base level training data set. We build one conflict resolver for each type of conflict. During implementation, we define a threshold  $\tau$ . We train a conflict resolver only when  $\epsilon > \tau$ . The intuition is to search for a conflict resolver that is able to have higher accuracy on that subset of data than the original stacking algorithm on the same subset. If we fail to find such a conflict resolver, we will use the original stacking algorithm on that subset.

An alternative approach is to add base classifiers to reduce conflicts. The addition of a base classifier may help distinguish data items that are confusing for the stacked generalizer. For bimodal problems, we can prove that adding a base classifier will at least not add more conflicts, implying that it may actually reduce conflicts. Consider the conflict pair:  $((c_1, c_2, \dots, c_t), 1)$  and  $((c_1, c_2, \dots, c_t), 0)$ . Assume  $\mathbf{T} = |((c_1, c_2, \dots, c_t), 1)|$  and  $\mathbf{S} = |((c_1, c_2, \dots, c_t), 0)|$ . Without loss of generality, suppose  $\mathbf{T} \leq \mathbf{S}$ .  $\mathbf{T}$  is the number of conflicts that will be mislabeled by the meta-classifier. When we

add a base classifier  $C_{t+1}$ , both  $((c_1, c_2, \dots, c_t), 1)$  and  $((c_1, c_2, \dots, c_t), 0)$  will be partitioned into two new types. The data associated with  $((c_1, c_2, \dots, c_t), 1)$  are divided into  $((c_1, c_2, \dots, c_t, 1), 1)$  and  $((c_1, c_2, \dots, c_t, 0), 1)$ . Similarly, the data associated with  $((c_1, c_2, \dots, c_t), 0)$  are divided into:  $((c_1, c_2, \dots, c_t, 0), 0)$  and  $((c_1, c_2, \dots, c_t, 1), 0)$ . The original conflict derives new conflicts with the addition of another base classifier:  $((c_1, c_2, \dots, c_t, 1), 1)$  and  $((c_1, c_2, \dots, c_t, 1), 0)$ ;  $((c_1, c_2, \dots, c_t, 0), 1)$  and  $((c_1, c_2, \dots, c_t, 0), 0)$ . We count their occurrences.  $\mathbf{t}_1 = |((c_1, c_2, \dots, c_t, 1), 1)|$ ,  $\mathbf{s}_1 = |((c_1, c_2, \dots, c_t, 1), 0)|$ ;  $\mathbf{t}_2 = |((c_1, c_2, \dots, c_t, 0), 1)|$ ,  $\mathbf{s}_2 = |((c_1, c_2, \dots, c_t, 0), 0)|$ . We know that  $\mathbf{t}_1 + \mathbf{t}_2 = \mathbf{T}$  and  $\mathbf{s}_1 + \mathbf{s}_2 = \mathbf{S}$ . One of the following must hold true, because  $\mathbf{T} \leq \mathbf{S}$ :

1.  $\mathbf{t}_1 \leq \mathbf{s}_1$  and  $\mathbf{t}_2 \leq \mathbf{s}_2$  (in which  $\mathbf{t}_1 + \mathbf{t}_2 = \mathbf{T}$  is the total number of conflicts of the two new conflict types)
2.  $\mathbf{t}_1 \leq \mathbf{s}_1$  and  $\mathbf{s}_2 \leq \mathbf{t}_2$  (in which  $\mathbf{t}_1 + \mathbf{s}_2 \leq \mathbf{T}$  is the number of conflicts)
3.  $\mathbf{s}_1 \leq \mathbf{t}_1$  and  $\mathbf{t}_2 \leq \mathbf{s}_2$  (in which  $\mathbf{s}_1 + \mathbf{t}_2 \leq \mathbf{T}$  is the number of conflicts)

This proves that adding a classifier will possibly reduce conflicts. Conversely, we can also show that removing any base classifier will not decrease any conflicts, which implies that it may actually increase conflicts.

In the above 3 conditions, the first one will not reduce any number of conflicts, it is still  $\mathbf{t}_1 + \mathbf{t}_2 = \mathbf{T}$ . In the second case or third case, we might reduce conflicts since  $\mathbf{t}_1 + \mathbf{s}_2 \leq \mathbf{T}$  or  $\mathbf{s}_1 + \mathbf{t}_2 \leq \mathbf{T}$ . Let us see what accuracy of the added classifier  $C_{t+1}$  is required to realize the reduction.

The accuracy of the added classifier on this subset of data is given by  $(\mathbf{t}_1 + \mathbf{s}_2)/(\mathbf{T} + \mathbf{S})$ . Recall that  $\mathbf{t}_1 = |((c_1, c_2, \dots, c_t, 1), 1)|$  and  $\mathbf{s}_2 = |((c_1, c_2, \dots, c_t, 0), 0)|$ . Also, the original stacking algorithm's accuracy on this subset is  $\mathbf{S}/(\mathbf{T} + \mathbf{S})$ . We have two choices here. One is to make  $\mathbf{t}_1 + \mathbf{s}_2 \leq \mathbf{T}$ . To do this, the added classifier's accuracy, which is given by  $(\mathbf{t}_1 + \mathbf{s}_2)/(\mathbf{T} + \mathbf{S})$ , will have to be much smaller than that of the original stacking (smaller than  $\mathbf{T}/(\mathbf{T} + \mathbf{S})$  rather than  $\mathbf{S}/(\mathbf{T} + \mathbf{S})$ ). Another way is to make  $\mathbf{s}_1 + \mathbf{t}_2 \leq \mathbf{T}$ . This is the same as making  $\mathbf{t}_1 + \mathbf{s}_2 \geq \mathbf{S}$ . This condition is the same as requiring the accuracy of the added classifier  $(\mathbf{t}_1 + \mathbf{s}_2)/(\mathbf{T} + \mathbf{S})$  to be bigger than that of stacking on this subset,  $\mathbf{S}/(\mathbf{T} + \mathbf{S})$ . To improve accuracy and reduce conflicts, we have two conditions: the accuracy of the added classifier must be either smaller than  $\mathbf{T}/(\mathbf{T} + \mathbf{S})$  or bigger than  $\mathbf{S}/(\mathbf{T} + \mathbf{S})$ . These two conditions are actually the same. Since for binary problems, flipping the prediction (from 1

to 0 and 0 to 1) will generate the other case. We are interested in searching for a classifier that is more accurate than the original stacking. The above reasoning is dualistic and so the condition is both sufficient and necessary. So here we also need a classifier that is more accurate than stacking on this subset of conflicting data. This is exactly the same condition that the conflict resolver, as discussed previously, has to satisfy in order to solve the problem. This means that the above two methods are actually identical since the possible solution for them will have to be the same. The only difference is how they are used. Therefore we need to look for one solution only. Here we consider the conflict resolver.

### 2.3 Recursive-Stacking

To improve stacking performance and to meet the desired accuracy condition, we next detail a recursive, tree-structured learning process. Empirical studies show that stacking usually has higher accuracy than any of the base classifiers. Our idea is to apply stacking recursively on each subset of data that generate conflicts to reduce conflicts and increase overall training accuracy. Figure 2 and Figure 3 show the outline of the algorithm. The idea is to use a *divide-and-conquer approach* to reduce the amount of conflicts. If the level of conflicts is too high in a set of examples, the set is split into smaller subsets, each of which corresponds to a conflict type. For each subset, new base classifiers are generated. That is, a set of new classifiers are learned to handle a particular conflict type. If the level of conflicts persists, the process is repeated.

*Recursive-Stacking* builds a tree that helps reduce conflicts. The algorithm is similar to building a decision tree (e.g. ID3), in which the training set is recursively partitioned from the root to the leaves. Each tree node has the result of a stacking episode (base classifiers only, the meta-classifier is not generated) and each branch corresponds to a conflict type. In place of the meta-classifier, a rote table stores the majority predicted classification for each conflict type. If the level of conflicts is too high at a node, examples are partitioned according to the conflict types and each child node has examples from one conflict type. At each child node, this process is repeated recursively until the conflict ratio ( $\epsilon$ ) is below a certain threshold ( $\tau$ ) or the number of examples at the node is too small for effective learning (called *stopping criteria* in the algorithm).

A node of the tree will have a subnode for type  $(c_1, c_2, \dots, c_t)$  if that conflict

**Algorithm: Recursive Stacking-Training****Input:**training set  $\mathcal{S}$ learning algorithms  $A_1, \dots, A_t$ CV-partitioning fold  $n$  ( $2 \leq n \leq |\mathcal{S}|$ )conflict ratio threshold  $\tau$ **Output:**

A Recursive-Stacking tree

**begin**

1. call stacking to generate meta-level training set  $\mathcal{M}_{train}$ ;
2. analyze  $\mathcal{M}_{train}$  for conflict types;
  - $\forall (c_1, c_2, \dots, c_t)$  conflict whose conflict ratio  $\epsilon \leq \tau$ , mark it as a *terminator conflict*
  - $\forall (c_1, c_2, \dots, c_t)$  conflict whose conflict ratio  $\epsilon > \tau$ 
    - if** *stopping criteria* is not met **then**
    - begin**
    - take all base level training data of type  $(c_1, c_2, \dots, c_t)$  into  $\mathcal{S}_{(c_1, c_2, \dots, c_t)}$
    - call **Recursive Stacking-Training** on  $\mathcal{S}_{(c_1, c_2, \dots, c_t)}$
    - end**
    - else**
    - mark this conflict type as a *terminator conflict*
3. return the Recursive-Stacking tree

**end**

Figure 2: Recursive Stacking Algorithm - Training

type is further resolved by the algorithm. The node is the *parent node* of the new subnode, and  $(c_1, c_2, \dots, c_t)$  of this parent node is called the *parent conflict* of the subnode. A node can have a maximum of  $2^t$  subnodes. If a conflict has no subnode, we call it a *terminator*. A leaf node in the tree is one that has all of its conflict types as terminators. A tree with a single root node only (depth = 0) is the original stacking except that we use a rote-table to record each conflict type and the majority label.

Figure 4 shows a simple example with 2 base classifiers. The root node identifies that conflicts (0,1) and (1,0) are confusing, since their  $\epsilon$  are 0.48 and 0.47 respectively. For both (0,1) and (1,0), Recursive-stacking generates a node to reduce conflicts.

When the computed tree is used for testing, an instance is supplied to the base classifiers at the root. If conflict type  $(c_1, c_2, \dots, c_t)$  predicted by the base classifiers is not a terminator, the instance is sent to the corresponding child conflict type based on the predictions of the base classifiers. When a leaf node is reached, based on the predictions from the leaf's base classifiers, the leaf's

**Algorithm: Recursive Stacking-Testing****Input:**testing Data:  $\mathcal{D}$ **Output:**

classifications

**begin**

1.  $\forall(\mathbf{x}_i, ?) \in \mathcal{D}$  call stacking to return meta-level testing item  $(C_1(\mathbf{x}), \dots, C_t(\mathbf{x}))$  and put it into  $\mathcal{M}_{test}$
2.  $\forall(\mathbf{x}_i, ?) \in \mathcal{D}$  put it into subset  $\mathcal{D}_{(c_1, c_2, \dots, c_t)}$  according to its type  $(c_1, c_2, \dots, c_t)$  in  $\mathcal{M}_{test}$ .
3. for each subset  $\mathcal{D}_{(c_1, c_2, \dots, c_t)}$ 
  - if**  $(c_1, c_2, \dots, c_t)$  is a terminator conflict **then**
  - $\forall(\mathbf{x}_i, ?) \in \mathcal{D}_{(c_1, c_2, \dots, c_t)}$  use the majority label of type  $(c_1, c_2, \dots, c_t)$
  - else**
  - call **Recursive Stacking - Testing** with  $\mathcal{D}_{(c_1, c_2, \dots, c_t)}$  and use the returned classifications
4. Combine predictions for each subset and return

**end**

Figure 3: Recursive Stacking Algorithm - Testing

rote table is used to generate the final prediction.

Recursive-Stacking is different from hierarchical stacking (a way of hierarchical meta-learning)[4]. In hierarchical stacking, stacking is applied on the meta-level training data. But in this new algorithm, we use a *divide-and-conquer* strategy by calling stacking on different subsets of the original training data. This is entirely new from the original stacking algorithm.

After learning, post-pruning is done on the tree to correct overfitted nodes and to remove redundant nodes. We first inspect every node to correct any that are not statistically significant to contribute to an accuracy increase. For a conflict type whose majority label is different from that of its parent conflict, if the  $\epsilon$  ratio of such a conflict is close to 0.5, it is doubtful that it will contribute to an increase in accuracy. Since the flip of the label from that of its parent may be due to overfitting of the data. All these kind of nodes ought to be "corrected" back to its parent's majority label.

A leaf node of a tree is redundant if the majority labels of all its conflict types are the same as that of its direct conflict parent. And a non-leaf node is redundant if all its subnodes are redundant. Redundant nodes will not improve accuracy since the decisions are the same as the parent conflict. We remove any such nodes from the tree in a depth first search manner after training.

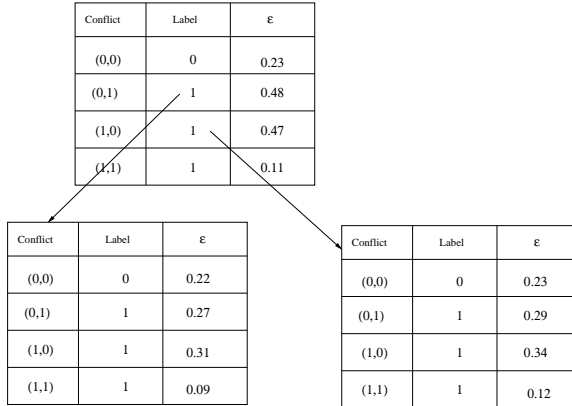


Figure 4: Sample Recursive Stacking Tree

There are other minor details about this algorithm left for a companion report [10].

### 3 Experiments and Results

We are interested in finding out answers to the following questions:

1. Will the Recursive-Stacking algorithm improve accuracy over stacking?
2. How deep will the tree grow before we can improve accuracy?
3. How deep will the tree grow before overfitting appears?
4. What are the simple approaches to correct possibly overfitted nodes?

#### 3.1 Experimental Set-up

In our experiments, four data sets were used: two artificial boolean data sets and two real world credit card transaction data sets. We used 2-fold to generate meta-level training data for different nodes of the tree. If the training data size is smaller than 200, the tree stops growing. We utilized a simple threshold value  $\pi$  to test the effectiveness of a new node. For any conflict type whose prediction label is different from that of its parent, if  $\epsilon \geq \pi$ , reverse the label

1. Results of RIPPER, ID3, CART on Boolean Data (5678)											
Base Learner	1	2	3	4	5	6	7	8	9	10	Avg
RIPPER	0.8004	0.7885	0.7958	0.7955	0.7916	0.7995	0.7992	0.7962	0.7869	0.7882	0.7942
CART	0.6866	0.6774	0.6820	0.6701	0.6832	0.6774	0.6848	0.6884	0.6804	0.6743	0.6805
ID3	0.6286	0.5935	0.6009	0.5981	0.6067	0.6063	0.6298	0.6280	0.6221	0.6160	0.6130
2. Recursive Stacking (uncorrected) on Boolean Data (5678)											
Depth	1	2	3	4	5	6	7	8	9	10	Avg
0	0.8004	0.7885	0.7958	0.7955	0.7916	0.7995	0.7992	0.7962	0.7869	0.7882	0.7942
1	0.8087	0.7925	0.8099	0.7989	0.7873	0.7965	0.8032	0.8050	0.7842	0.7869	0.7973
2	0.8087	0.7922	0.8099	0.8007	0.7873	0.7974	0.8032	0.8065	0.7842	0.7869	0.7977
3. Results of Recursive Stacking (corrected) on Boolean Data (5678)											
Depth	1	2	3	4	5	6	7	8	9	10	Avg
0	0.8004	0.7885	0.7958	0.7955	0.7916	0.7995	0.7992	0.7962	0.7869	0.7882	0.7942
1	0.8065	0.7937	0.7989	0.7992	0.7937	0.8001	0.8090	0.8026	0.7900	0.7903	0.7984
2	0.8065	0.7937	0.7989	0.7992	0.7937	0.8001	0.8090	0.8026	0.7900	0.7903	0.7984

Figure 5: Results on Boolean Data (5678)

back to that of its parent. For experiments on all 4 sets of data, we fixed  $\pi$  to be 0.4.

We used RIPPER, CART, ID3 and C4.5 as the base learners. RIPPER was obtained from William Cohen[6]. CART, ID3 and C4.5 are part of the IND package[3].

### 3.2 Boolean Artificial Data Set

We generated an artificial data set of 15 boolean variables. The data item is 1 if 5, 6, 7 or 8 variables out of 15 are 1. We call this set Boolean.5678. Since there are 15 variables, there are  $2^{15} = 32768$  data items in total. The percentage of data items with label 1 are about 64%. We do not replicate any data items in our test. The base level algorithms are RIPPER, ID3 and C4.5. 10 fold CV is applied on the data to average our results. Since no data items are replicated, any examples in the testing set are not in the corresponding training set. Figure 5 displays the results. RIPPER is the most accurate base classifier. The average accuracy of RIPPER is 0.7942. Both ID3 and CART are significantly worse than RIPPER, their overall accuracy are all below 0.70. CART’s accuracy is better than that of ID3.

The next two tables show the results for Stacking and Recursive Stacking. The second table is the results for the case where we only applied the strategy of removing nodes. The third table displays results for the case where we applied both strategies of removing and correcting nodes. As we compare

1. Results of RIPPER, ID3, CART on Boolean Data (45)											
Base Learner	1	2	3	4	5	6	7	8	9	10	Avg
RIPPER	0.8703	0.8627	0.8663	0.8825	0.8724	0.8685	0.8685	0.8666	0.8764	0.8758	0.8710
CART	0.8666	0.8666	0.8666	0.8666	0.8666	0.8666	0.8666	0.8666	0.8669	0.8669	0.8667
ID3	0.7812	0.7708	0.7711	0.7855	0.7763	0.7797	0.7638	0.7885	0.7769	0.7827	0.7776
2. Recursive Stacking (uncorrected ) on Boolean Data (45)											
Depth	1	2	3	4	5	6	7	8	9	10	Avg
0	0.8511	0.8419	0.8425	0.8486	0.8499	0.8569	0.8489	0.8666	0.8510	0.8556	0.8513
1	0.8770	0.8688	0.8688	0.8819	0.8682	0.8734	0.8789	0.8666	0.8797	0.8819	0.8745
2	0.8746	0.8822	0.8651	0.8856	0.8715	0.8773	0.8746	0.8706	0.8852	0.8840	0.8771
3. Results of Recursive Stacking (corrected) on Boolean Data (45)											
Depth	1	2	3	4	5	6	7	8	9	10	Avg
0	0.8511	0.8419	0.8425	0.8486	0.8499	0.8569	0.8489	0.8666	0.8510	0.8556	0.8513
1	0.8770	0.8688	0.8688	0.8819	0.8682	0.8734	0.8789	0.8666	0.8797	0.8819	0.8745
2	0.8746	0.8822	0.8651	0.8856	0.8715	0.8773	0.8746	0.8706	0.8852	0.8840	0.8771

Figure 6: Results on Boolean Data (45)

results of the original stacking (tree depth = 0) and recursive stacking in both the second and third tables, recursive stacking (tree depth  $\geq 1$ ) consistently beats the overall accuracy of stacking for all 10 runs of the experiment. The increase is consistently above 0.004 (or 0.4%). By comparing the results of original stacking and those of RIPPER, we see that the results are exactly the same for all 10 runs. This means that original stacking simply picks RIPPER’s output, and it has no increase of accuracy over the base classifiers (see Section 2.1 and Figure 1). However, recursive stacking has a consistent increase of 0.5%. We are interested to find out the tendency of accuracy increase with the growth of tree depth. From the second table, when the depth of the tree grows from 0 to 2, the overall accuracy steadily increases from 0.7942 to 0.7984. And we also see that the biggest increase takes place when the tree depth grows from 0 to 1 or from original stacking to recursive stacking. When the tree grows deeper, there is less room for improvement, so the increase rate drops. We are interested to see if the increase in accuracy by Recursive Stacking is stable. For 10 runs, 5 showed recursive stacking does better, and the other 5 showed slightly worse performance. But when correcting is used, the results are always better throughout the 10 runs.

We generated another data set: Boolean.45. The data item is 1 when 4 or 5 variables are 1. There are 13.3% data items in the training set with label 1. This set is relatively skewed. Figure 6 lists the results. There are a few interesting observations here. One is that stacking does not do better than either

1. Results of RIPPER, CART and C4.5 on First Union Credit Card Data						
Base Learner	1	2	3	4	5	Avg
RIPPER	0.9448	0.9494	0.9489	0.9438	0.9495	0.9475
CART	0.9470	0.9508	0.9511	0.9471	0.9494	0.9486
C4.5	0.9484	0.9498	0.9519	0.9466	0.9511	0.9493
2. Recursive Stacking (uncorrected ) on First Union Credit Card Data						
Depth	1	2	3	4	5	Avg
0	0.9522	0.9568	0.9566	0.9523	0.9557	0.9544
1	0.9508	0.9555	0.9573	0.9533	0.9562	0.9544
2	0.9507	0.9550	0.9573	0.9530	0.9548	0.9540
3. Results of Recursive Stacking (corrected) on First Union Credit Card Data						
Depth	1	2	3	4	5	Avg
0	0.9522	0.9568	0.9566	0.9523	0.9557	0.9544
1	0.9525	0.9577	0.9576	0.9532	0.9565	0.9551
2	0.9523	0.9573	0.9576	0.9533	0.9555	0.9549

Figure 7: Results on First Union Credit Card Data

RIPPER or CART. Consider the predictions on the testing set and the conflicts listed in Figure 1. The reason is that for types (1,0,0) and (1,0,1) (whose  $\epsilon$  are 47.4% and 39.6% respectively), the majority label picked by stacking turns out to be a minority for the testing data. But Recursive-Stacking outperforms all the base classifiers. The accuracy increase by applying Recursive-Stacking over pure stacking is consistently more than 2.3%. Note, the post-learning correction has no effect at all, since the  $\epsilon$  of non-root nodes are below that threshold.

For the above data sets, we didn't see any overfitting for the maximal depth=2 used in the experiments.

### 3.3 Credit Card Data Set

We also tested Recursive-Stacking on two real world data sets: First Union Credit Card Transaction Data and Chase Credit Card Transaction Data. These data sets were provided for our research in fraud and intrusion detection. The classification of the data is either legitimate or fraudulent. For a detailed description of the data schema, refer to [14].

From each bank, we obtained 0.5 million data spanning a whole year. The First Union data was not uniformly sampled for each month. The percentage of fraud ranges from 4% to over 20% over each month and the size of data for each month varies. We have done some initial experiments. Due to the non-

1. Results of RIPPER, CART and C4.5 on Chase Credit Card Data											
Base Learner	1	2	3	4	5	6	7	8	9	10	Avg
RIPPER	0.8087	0.8891	0.8758	0.8869	0.8890	0.8796	0.8970	0.9013	0.8640	0.8496	0.8741
CART	0.8861	0.8906	0.8760	0.8878	0.8899	0.8810	0.9002	0.9035	0.8723	0.8756	0.8863
C4.5	0.8510	0.8720	0.8630	0.8708	0.8726	0.8623	0.8817	0.8887	0.8579	0.8621	0.8682
2. Recursive Stacking (uncorrected) on Chase Credit Card Data											
Depth	1	2	3	4	5	6	7	8	9	10	Avg
0	0.8758	0.8934	0.8788	0.8918	0.8932	0.8848	0.9022	0.9047	0.8730	0.8778	0.8875
1	0.8817	0.8928	0.8776	0.8858	0.8936	0.8831	0.9010	0.9022	0.8713	0.8761	0.8865
2	0.8808	0.8926	0.8831	0.8857	0.8932	0.8831	0.9007	0.9018	0.8708	0.8761	0.8868
3. Results of Recursive Stacking (corrected) on Chase Credit Card Data											
Depth	1	2	3	4	5	6	7	8	9	10	Avg
0	0.8758	0.8934	0.8788	0.8918	0.8932	0.8848	0.9022	0.9047	0.8730	0.8778	0.8875
1	0.8771	0.8942	0.8790	0.8918	0.8932	0.8850	0.9022	0.9045	0.8729	0.8785	0.8878
2	0.8769	0.8942	0.8794	0.8918	0.8932	0.8850	0.9022	0.9041	0.8728	0.8785	0.8878

Figure 8: Results on Chase Credit Card Data

uniform distribution of data in each month, the models learned from different months differ greatly. In our experiment, we removed all date and other fields that are not available at authorization and then culled all transactions into one large data set. The results reported in Figure 7 were produced from this conditioned data set. We partitioned the data into 10 folds, used one fold for training and another fold for testing for a total of 5 runs. Each fold is disjoint. We didn't use conventional CV partitioning, because the size of the data is huge, it takes a long time to learn a classifier. The results we have obtained on this real world data set are very close to the results on the boolean artificial data sets. Stacking improves the accuracy of the best base classifier by 0.7% and Recursive stacking improves stacking by 0.1%. We have seen overfitting in 2 out of the 5 runs without using post-learning correction. We also see that Recursive-Stacking with correction outperforms stacking consistently over all 5 runs.

The Chase Credit Card data was more uniformly sampled than First Union Data. The fraud percentage of each month ranges from 17% to 23%. The data set size of each month varies from 28k to 50K. We believe that this data set is more appropriate to try our algorithm in a real world situation. In our experiment, we used data of one month for training and data of 2 months later for testing. (In a real world content, there is a one month billing cycle and a one month investigation to ultimately determine if a transaction is fraudulent.) Since our data set is 12 months, only 10 experiments are feasible. Figure 8

shows the results of the experiment. On the average, Recursive-Stacking with correction improves the accuracy of stacking by 0.03%. The best improvement is in the first run where the increase is 0.06%. In some cases, we see that it overfits at the second tree level. The reason appears to be that the data sets from different months are not uniform in both size and fraud distribution. The result of stacking may already be very close to the best we can obtain, which does not leave much room for improvement. We ran Boosted Naive Bayes[8] on the same data sets with up to 10 rounds of boosting. The result by boosting and stacking are very close on average.

## 4 Discussion

Combining a large number of classifiers in Recursive Stacking will be a problem since the number of conflict types is  $2^t$ . We propose two solutions. One method is to apply hierarchical stacking. We may group a small number of base classifiers together and apply the Recursive-Stacking algorithm. We can then combine each together using stacking/Recursive-Stacking. Another way is to combine all the base classifiers together but we prune any conflict types that are insignificant (whose number is only a small percentage of meta-level data size). To do this, we use an ID3-like algorithm to learn the meta-level data. Each leaf of such a tree is a conflict type. We remove any leaf nodes in the tree that are not significant and use its parent node's decision. Future experiments will help establish whether either of these methods are effective.

In our experiment, we used simple a threshold  $\pi = 0.4$  to correct any nodes whose  $\epsilon > \pi$ . This approach worked well for all 4 data sets under study. We looked at the predictions by Recursive-Stacking on the testing data. There is a general tendency that a node with big training size and small  $\epsilon$  is more likely to be stable and have high accuracy gains. Future experiments will consider the sensitivity of these thresholds on the overall accuracy gain.

## 5 Conclusion

The presence of conflicts in a stacked generalizer is a fundamental problem limiting improvements in accuracy. The proposed Recursive-Stacking attempts to resolve conflicts. It has been shown to increase accuracy of stacking by as

much as 3% under the data sets in our study. Accuracy increase is nominal when the tree is only at a depth of 1. The cost to learn such a shallow tree with depth of 1 and 2-fold CV partition is affordable. This algorithm is scalable to allow the stacking of a large number of classifiers via hierarchical learning and pruning strategy.

## References

- [1] K. Ali. On Explaining Degree of Error Reduction due to Combining Multiple Decision Trees. *Integrating Multiple Learning Model Workshop*, AAAI-96, Portland, Oregon.
- [2] C. Brodley and T. Lane. Creating and Exploiting Coverage and Diversity. *Integrating Multiple Learning Model Workshop*, AAAI-96, Portland, Oregon.
- [3] W. Buntine and R. Caruana. Introduction to IND and Recursive Partitioning, NASA Ames Research Center.
- [4] P. Chan. An Extensible Meta-learning Approach for Scalable and Accurate Inductive Learning. *Ph.D. Thesis, Department of Computer Science, Columbia University, New York, New York*, 1996.
- [5] R.T. Clemen. Combining Forecasts: A Review and Annotated Bibliography. *International Journal of Forecasting*, 5:559-583.
- [6] W. Cohen. Fast Effective Rule Induction. In *Proc. Twelfth Intl. Conf. on Machine Learning*. Morgan Kaufman.
- [7] T. Dietterich. Machine Learning Research: Four Current Directions. *submitted for publication*, 1997.
- [8] C. Elkan. Boosted Naive Bayes. *submitted for publication*, 1997.
- [9] D. Fan, P. Chan and S. Stolfo. A Comparative Evaluation of Combiner and Stacked Generalization. *Integrating Multiple Learning Model Workshop*, AAAI-96, Portland, Oregon.
- [10] D. Fan, P. Chan and S. Stolfo. On different aspects of Recursive-Stacking. *to be submitted*.
- [11] S. Hashem. Optimal Linear Combination of Neural Networks. *Phd Thesis, Purdue University, School of Industrial Engineering, Lafayette, IN*, 1993.
- [12] M.I. Jordan and R.A. Jacob. Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation*, 6(2), p.181-p.214.
- [13] S. Stolfo, A. Prodromidis, S. Tselepis, W. Lee, D. Fan and P. Chan. JAM: Java Agents for Meta-learning over Distributed Databases. In *Prod. Third Intl. Conf. Knowledge Discovery and Data Mining*, 1997.
- [14] S. Stolfo, D. Fan, W. Lee, A. Prodromidis and P. Chan. Credit Card Fraud Detection Using Meta-learning: Issues and Initial Results. In *Working Notes AAAI-97*, 1997.
- [15] D. Wolpert. Stacked Generalization. *Neural Networks*, 5(2), p.241-p.260.