A Combinatorial Fusion Method for Feature Construction

Ye Tian[1], **Gary M. Weiss**[2], **D. Frank Hsu**[3], **and Qiang Ma**[4]
[1]Department of Computer Science, New Jersey Institute of Technology, Newark, NJ, USA
[2, 3]Department of Computer and Information Science, Fordham University, Bronx, NY, USA
[4]Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Abstract - *This paper demonstrates how methods borrowed from information fusion can improve the performance of a classifier by constructing (i.e., fusing) new features that are combinations of existing numeric features. The new features are constructed by mapping the numeric values for each feature to a rank and then averaging these ranks. The quality of the fused features is measured with respect to how well they classify minority-class examples, which makes this method especially effective for dealing with data sets that exhibit class imbalance. This paper evaluates our combinatorial feature fusion method on ten data sets, using three learning methods. The results indicate that our method can be quite effective in improving classifier performance.*

Keywords: Feature construction, class imbalance, information fusion

1 Introduction

The performance of a classification algorithm is highly dependent on the descriptions associated with the example. For this reason, good practitioners will choose the features used to describe the data very carefully. However, deciding which information to encode and how to encode it is quite difficult and the best way to do so depends not only on the domain, but on the learning method. For this reason, there have been a variety of attempts over the years to automate part of this process. This work has had a variety of names over the years (although sometimes the emphasis is different) and has been called constructive induction [13], feature engineering [17], feature construction [6] and feature mining [11]. In this paper we discuss how existing numerical features can be combined, without human effort, in order to improve classification performance.

The work described in this paper is notable for several reasons. First, unlike the majority of work in this area, we are specifically concerned with improving the performance of data with substantial class imbalance. Such problems are challenging but quite common and are typical in domains such as medical diagnosis [7], fraud detection [4], and failure prediction [19]. Furthermore, there are reasons to believe that this important class of problems has the most to benefit from feature construction, since some learners may not be able to detect subtle patterns that only become apparent when several features are examined together [18]. Our work also differs from other work in that our feature combination operator does not directly use the values of the component features but rather their ranks. This allows us to combine numerical features in a meaningful way, without worrying about issues such as scaling. This approach is particularly appropriate given the increased interest in the use of ranking in the data mining [10] and machine learning communities [5]. Our approach also can be viewed as an extension of work from the information fusion community, since techniques similar to the ones we use in this paper have been used to "fuse" information from disparate sources [9]. The work in this paper can be viewed as a specific type of information fusion, which we refer to as feature fusion.

We describe our combinatorial feature-fusion method in detail in Section 2 and then describe our experiments in Section 3. The results from these experiments are described and analyzed in Section 4. Related work is discussed in Section 5. Our main conclusions and areas for future work are then described in Section 6.

2 Combinatorial Feature Fusion

This section describes the basic combinatorial feature-fusion method. We introduce relevant terminology and describe some of the basic steps employed by the feature fusion method. We then describe some general schemes for fusing features and end the section with a detailed description of our combinatorial feature fusion algorithm.

2.1 Terminology and Basic Steps

In this section we will use a simple example to explain the relevant terminology and preliminary steps related to feature fusion. This example will also be used later in this Section to help explain the feature-fusion algorithm. Because our feature-fusion method only works with numeric features, for simplicity we assume all features are numeric. Non-numeric features are not a problem in practice—they simply will be passed, unaltered, to the classifier.

A data set is made up of examples, or records, each of which has a fixed number of features. Consistent with previous work on information fusion [9, 10] we view the value of a feature as a *score*. Typical examples of scores are a person's salary, a student's exam score, and a baseball pitcher's earned run average. In the first two cases a higher score is desirable but in the last case a lower one is preferable.

Table 1 introduces a sample data set with eight examples, labeled A-H, with five numeric features, F1-F5, and a binary class variable. In this example class 1 is the minority class and comprises 3/8 or 37.5% of the examples.

TABLE 1
A SAMPLE DATASET

	F1	F2	F3	F4	F5	Class
A	1	4	3	2	8	1
B	3	3	5	5	4	0
C	5	5	2	6	7	1
D	7	6	15	3	2	0
E	11	13	16	7	14	0
F	15	16	4	13	11	0
G	9	7	14	1	18	1
H	17	15	9	8	3	0

Early in our combinatorial feature-fusion method we replace each score with a rank, where a lower rank is better. We convert each score into a rank using a *rank function*, which adheres to the standard notion of a rank. We sort the score values for each feature in either increasing or decreasing order and then assign the rank based on this ordering. Table 2 shows the values of the features for the sample data set after the scores have been replaced by ranks, where the ranks were assigned after sorting the feature values in increasing order. As a specific example, because the three lowest values for F3 in Table 1 are 2, 3, 4 and these values appear in rows C, A, and F, respectively, the ranks in Table 2 for F3 for these records are 1, 2, and 3, respectively.

TABLE 2
SAMPLE DATASET WITH SCORES REPLACED BY RANKS

	F1	F2	F3	F4	F5
A	1	2	2	2	5
B	2	1	4	4	3
C	3	3	1	5	4
D	4	4	7	3	1
E	6	6	8	6	7
F	7	8	3	8	6
G	5	5	6	1	8
H	8	7	5	7	2

We determine whether the ranks should be assigned based on increasing or decreasing order of the score values by determining the performance of the feature using both ordering schemes and selecting the ordering that yields the best performance (we describe how to compute a feature's performance shortly). In our method, once the scores are replaced with a rank the scores are never used again. The rank values are used when combining features and are the features values that are passed to the learning algorithm.

Next we show how to compute the "performance" of a feature. This performance metric essentially measures how well the rank of the feature correlates with the minority-class examples. That is, for a feature, do the examples with a good rank tend to belong to the minority class? We explain how to compute this performance metric using feature F2 from the sample data set. First we sort the records in the data set by the rank value of F2. The results are shown in Table 3. The performance of F2 is then computed as the fraction of the records at the "top" of the table that belong to the minority class. The number of "top" records that we examine is based on the percentage of minority-class examples in the training data. In this case 3 of 8 of the training examples (37.5%) belong to the minority class so we look at the top 3 records. In this example that means that the performance of F2 is 2/3, since two of the three class values for these records is a "1", which is the minority-class value. Given this scheme, the best performance value that is achievable is 1.0.

TABLE 3
RANKED LIST FOR F2

	F2 Rank	Class
B	1	0
A	2	**1**
C	3	**1**
D	4	0
G	5	1
E	6	0
H	7	0
F	8	0

We may similarly compute the performances for all of the individual features. Table 4 shows that for this simple example F1–F4 all have performances of 2/3 and F5 has a performance of 0.

TABLE 4
PERFORMANCE VALUES FOR ORIGINAL FEATURES

Feature	Performance
F1	0.67
F2	0.67
F3	0.67
F4	0.67
F5	0.00

This method is also used to compute the performance of the "fused" features. To do this we need to first determine the rank of a fused feature, so we can sort the examples by this rank. We compute this using a *rank combination function* that averages the ranks of the features to be combined. This is done for each record. As an example, suppose we want to fuse features F1–F5 and create a new feature, F1F2F3F4F5, which we will call F6. Table 5 shows the rank values for F6 for all eight records. The value for F6 for record A is computed as: (Rank(F1) + Rank(F2) + Rank(F3) + Rank(F4) + Rank(F5))/5 = (1+2+2+2+5)/5 = 2.4. We see that for this new feature, record "A" has the best (lowest) rank. Given these values, one can now compute the performance of the feature F6. Note that even though the values in Table 5 are not integers we can still consider them ranks. In order to compute the performance of F6, we only need to be able to sort by these values.

TABLE 5
RANK VALUES FOR F6 (F1F2F3F4F5)

	F6		F6
A	2.4	E	6.6
B	2.8	F	6.4
C	3.2	G	5.0
D	3.8	H	5.8

2.2 Combinatorial Fusion Strategies

The previous section introduced the terminology and basic steps required by our combinatorial fusion algorithm, but did not discuss how we decide *which* features to fuse. We discuss that topic in this section.

There are many possible strategies for choosing features to "fuse." In this paper we consider combinatorial strategies that look at all possible combinations or more restrictive variants that look at subsets of these combinations. Let *n* equal the number of numeric features available for combination. To look at all possible combinations would require that we try each single feature, all pairs of features, all triples, etc. The total number of combinations therefore equals C(*n*,1) + C(*n*,2) + … C(*n*, *n*), which equals $2^n - 1$. We refer to such a combinatorial fusion strategy as a fully-exhaustive fusion strategy.

We consider more restrictive variants of the fully-exhaustive fusion strategy because, depending on the value of *n*, this strategy may not be practical. The *k-exhaustive* fusion strategy will create all possible combinations using *k* of the *n* ($k < n$) numeric features. For example, a 6-exhaustive strategy for a data set with 20 numeric features will select 6 features and then fuse them in all possible ways, reducing the number of feature combinations by a factor of 2^{14}. In our algorithm we choose the subset of *k* features based on the performance values for the features, such as the ones in Table 6. Because it will not be expensive to include all of the original features, we always include the $n - k$ original features. The 6-exhaustive fusion strategy is one of the three strategies analyzed in this paper.

The k-exhaustive fusion strategy trades off a reduced number of features for the ability to fully combine these features. In some cases it may be better to involve more features in the fusion process, even if they cannot be fused in all possible ways. The *k-fusion* strategy will use all *n* numeric features, but the length of the fused features is limited to length at most *k*. Thus if we have a data set with 20 numeric features and employ 2-fusion, all possible combinations of single features and pairs of features will be generated. This would yield C(20,1) + C(20,2) = 20 + 190 = 210 features. Similarly, 3-fusion would consider C(20,1) + C(20, 2) + C(20, 3), or 1140 feature combinations.

Table 6 shows the number of features generated by the different fusion strategies. In all cases, as stated before, all original features are included. Some cells are empty since $k \leq n$. If $k = n$ then the value computed is displayed in bold and corresponds to the fully-exhaustive strategy. Table 6 demonstrates that, given a limit on the number of features we can evaluate, we have a choice of fusion strategies. For example, given ten numeric features, one can use all ten features and generate combinations of length four, which would generate 385 features, or instead select the seven best ones and then fuse those in all possible ways (i.e., up to length 7), which would generate about 127 features (actually 130 when the three original features are included).

TABLE 6
COMBINATORIAL FUSION TABLE

Number Features	k-fusion for values of k shown below									
	1	2	3	4	5	6	7	8	9	10
1	**1**									
2	2	**3**								
3	3	6	**7**							
4	4	10	14	**15**						
5	5	15	25	30	**31**					
6	6	21	41	56	62	**63**				
7	7	28	63	98	119	126	**127**			
8	8	36	92	162	218	246	254	**255**		
9	9	45	129	255	381	465	501	510	**511**	
10	10	55	175	385	637	847	967	1012	1022	**1023**

2.3 The Combinatorial Fusion Algorithm

We now describe the algorithm for performing the combinatorial fusion. This algorithm is summarized in Table 7. We explain this algorithm by working through an example based on the data set introduced in Table 1.

For this example, we will use the 5-exhaustive strategy, so that we select the five best performing features and then fuse them in all possible ways. On line 1 of the algorithm we pass into the *Comb-Fusion* function the data, the features, a *k* value of 5 and a value of True for the Exhaustive flag. The next few steps were already described in Section 2.1. We convert the scores to ranks (line 3) and then calculate the performance of the original (unfused) features in the loop from lines 4-6. Then in lines 7-11 we determine which features are available for fusion. Since the Exhaustive flag is set, we restrict ourselves to the *k* best features (otherwise all features are available although they then may not be fused in all possible ways).

TABLE 7
THE FEATURE-FUSION ALGORITHM

```
1.  Function Comb-Fusion (Data, Features, k, Exhaustive)
2.  {
3.    ConvertScoresToRanks(Data, Features);
4.    for (f=1, f ≤ length(Features) , f++){
5.        Perf[f]=CalculatePerformance(f);
6.    }
7.    if (Exhaustive == TRUE) {
8.        FeaturesForFusion = best k features from Perf[];
9.    } else {
10.       FeaturesForFusion = Features;
11.   }
12.   New = FuseFeatures(FeaturesForFusion, k, Exhaustive);
13.   for (f=1, f ≤ length(New) , f++){
14.       CalculateRank(f);
15.       Perf2[f]=CalculatePerformance(f);
16.   }
17.   Sort(Perf2);
18.   Candidates = Perf2.features;
19.   // We now build up the final feature set
20.   Keep = Features;   // always use original features
21.   partition(Data, *TrainValid, Test);
22.   for (f in Candidates)
23.   {
24.       for (run=1; run ≤ 10, run++)
25.       {
26.           partition(TrainValid, *Training, *Validation);
27.           classifier = build-classifier(Training, Keep);
28.           PerfWithout[run] = evaluate(classifier, Validation);
29.           cand = pop(Candidates);
30.           classifier=build-classifier(Training, Keep ∪ cand);
31.            PerfWith[run] = evaluate(classifier, Validation);
32.       }
33.       if ( average(PerfWith[ ]) > average(PerfWithout[ ]) )
34.       {
35.            pval = t-test(PerfWith[], PerfWithout[]);
36.            if (pval ≤ .10) {
37.                Keep = Keep ∪ cand;
38.            }
39.       }
40.   }  // end for (f in Candidates)
41.   final-classifier = build-classifier(Training, Keep);
42.   final-performance = evaluate(Test, Keep);
43.  } // end Function Comb-Fusion
```

The actual generation of the fused features occurs on line 12. In this case, the five best features in *FeaturesForFusion* will be combined in all possible ways (in this example there are only five features to begin with). Given our decision to always include the original features to the classifier, the original features need not be returned by *FuseFeatures* (they are handled later on line 20).

Next, on lines 13-16 we calculate the rank for each fused feature and then calculate their performance. This is essentially the same steps that were done earlier for the original features. We then sort the features by decreasing performance value (line 17) and extract the features from this sorted list and save them (line 18) in *Candidates*, the ordered list of candidate fused features. The results for the best 14 performing fused features for our simple example are shown in Table 8. In this case *Candidates* equals {F3F4, F1F2, F1F3, …}.

TABLE 8
PERFORMANCE VALUES FOR 5-EXHAUSTIVE STRATEGY

Priority	Feature	Perf.	Priority	Feature	Perf.
1	F3F4	1	8	F1F2F4	0.67
2	F1F2	0.67	9	F1F3F4	0.67
3	F1F3	0.67	10	F1F3F5	0.67
4	F2F3	0.67	11	F2F3F4	0.67
5	F2F4	0.67	12	F3F4F5	0.67
6	F3F5	0.67	13	F1F2F3F4	0.67
7	F1F2F3	0.67	14	F1F2F3F5	0.67

In the second half of the algorithm, starting at line 19, we decide which of the *Candidate* features to include in the final feature set. We begin by initializing *Keep* to the set of original features. We then partition the data (line 21) into one set to be used for training and validation and another for testing. Beginning on line 22 we iterate over all of the fused features in the *Candidate* set.

A key question is how we determine when to add a feature. Even though a feature has a good performance score, it may not be useful. For example, the information encoded in the feature may be redundant with the features already included in the feature set. We adopt a pragmatic approach and only add a feature if it improves classifier performance on the validation set and the improvement is statistically significant. To determine this, within this main loop in the second half of the algorithm (lines 22 – 40) we execute ten runs (lines 24 – 32), repeatedly partitioning the training data into a training set and a validation set (line 26). If, averaged over the 10 runs (line 33) the classifier generated with the candidate feature (line 30) outperforms the classifier generated without it (line 28) and the p-value returned by the t-test (line 35) is ≤ .10 (line 36), then we add the feature to *Keep* (line 37). A p-value ≤ .10 means that we are 90% confident that the observed improvement reflects a true improvement in performance. In steps 41 and 42 we build the final classifier and evaluate it on the test set.

We should point out a few things. First, the actual implementation is more efficient in that we only need to build one classifier in the main loop, since the classifier from the previous iteration, and its performance, is still available. Similarly, we do not need to rebuild the classifier as indicated on line 41. Also, the performance of the classifier can be measured using either AUC or accuracy, and we use both measures in our experiments.

Table 9 shows the behavior of our simple example as each feature is considered. We only show the performance for the first 3 features. The last column indicates the feature being considered and a "+" indicates that it is added while the lack of this symbol indicates that it is not added because the conditions on lines 33 and 36 are not both satisfied. Each row corresponds to an iteration of the main loop starting at line 22 in the algorithm. The first row is based on the classifier built from the original feature set, containing features F1-F5. Note that the first and third features that are considered are added, because they show an improvement in AUC and the p-value is ≤ .10. As we add features we also measure the performance of each classifier on the test set, although this is not used in any of the decision making. The AUC for the test set at the end is reported, however. If we stopped the algorithm after the three iterations, we can conclude that the performance improved from an AUC of .682 to .774. It is of course critical not to use the test set results to determine whether to add a feature (and we do not).

TABLE 9
THE EXECUTION OF THE ALGORITHM ON A SIMPLE EXAMPLE

AUC		p-value	Feature
valid	test		(+ means added)
0.670	0.682	--	{F1,F2,F3,F4,F5}
0.766	0.757	0.001	**+F3F4**
0.731			F1F2
0.771	0.774	0.063	**+F1F3**

3 Description of Experiments

In this section we describe the datasets employed in our empirical study, the three learning methods that are utilized, and the methodology we use to conduct our experiments.

Table 10 describes the ten data sets used in our study. Note that the data sets are ordered in terms of decreasing class imbalance. The data sets come from several sources. The hepatitis, bands, income and letter-a data sets were obtained from the UCI machine learning repository [14], the crx data set was provided in the Data directory that came with the C4.5 code, the physics and bio data sets are from the 2004 KDD CUP challenge, the stock data set was provided by New York University's Stern School of Business, and the boa1 data set was obtained from researchers at AT&T.

TABLE 10
THE DATA SETS

Dataset Name	% Minority Class	Number Features	Dataset Size
protein+	0.59	14	20,000
letter-a*	3.9	15	20,000
income*+	5.9	12	10,000
stock*+	9.9	27	7,112
hepatitis*	19.8	12	500
physics+	24.9	8	20,000
german*	30.0	19	1,000
crx*+	44.1	5	450
bands*+	42.2	13	538
boa1+	49.8	25	5,000

In order to simplify the presentation and the analysis of our results, data sets with more than two classes were mapped to two-class problems. This was accomplished by designating one of the original classes, typically the least frequently occurring class, as the minority class and then mapping the remaining classes into the majority class. The data sets that originally contained more than two classes are identified with an asterisk (*). The letter-a data set was generated from the letter-recognition data set by making the letter "a" the minority class. Because we are only employing feature fusion for the numeric features, we deleted any non-numeric features from the data sets. While this is not necessary, since our method could just ignore the non-numeric fields, we did this so that we could better determine the impact of the feature fusion method. The data sets that had any non-numeric features are identified with a "+".

All of the learning methods that we use in this paper come from the WEKA data mining software [12]. The three learning methods that we use are Naïve Bayes, decision trees and 1-nearest neighbor. The decision tree algorithm is called J48 in WEKA and is an implementation of the C4.5 algorithm. The 1-nearest neighbor algorithm is referred to as IB1 in WEKA.

The experiments in our study apply a combinatorial feature-fusion strategy to each of the ten data sets listed in Table 10 and then record the performance with and without the fusion strategy. This performance is measured in terms of the area under the ROC curve (AUC), because ROC analysis [3] is a more appropriate performance metric than accuracy when there is class imbalance. Nonetheless we repeat some of our experiments with accuracy as the performance metric, since doing so it quite straightforward and accuracy is a very commonly used performance metric. The three combinatorial fusion strategies that are evaluated are the 2-fusion, 3-fusion and 6-exhaustive fusion strategies described in Section 2. In this study we utilize the three learning algorithms listed in Section 3 in order to see how the feature-fusion method benefits each algorithm. In the algorithm in Table 7 the data is partitioned such that 50% is used for training, 20% for validation, and 30% for testing.

4 Results

In this section we describe our main results. Because we are interested in improving classifier performance on data sets with class imbalance, and because of the known deficiencies with accuracy as a performance metric [16], we use AUC as our main performance measure. These AUC results are summarized in Table 11. The results are presented for ten data sets using the Naïve Bayes, decision tree, and 1-NN learning methods. Three combinatorial fusion strategies are evaluated: 2-Fusion (2F), 3-fusion (3F) and 6-Exhaustive (6EX). The AUC results are presented first without (w/o) and then with (w) the combinatorial fusion strategy. The "diff" column shows the absolute *improvement* in AUC resulting from the combinatorial fusion strategy, with negative values indicating that combinatorial fusion degraded the performance.

TABLE 11
AUC IMPROVEMENT WITH COMBINATORIAL FUSION

Dataset	Strat.	Bayes			Decision Trees			1-NN		
		w/o	w	Diff	w/o	w	Diff	w/o	w	Diff
bio	2F		.923	-.020		.752	.256		.663	.164
	3F	.943	.954	.010	.496	.742	.247	.499	.651	.152
	6EX		.926	-.017		.759	.264		.663	.164
letter-a	2F		.963	.000		.943	.021		.961	.024
	3F	.963	.960	-.003	.922	.919	-.003	.937	.937	.000
	6EX		.962	-.001		.937	.014		.961	.024
income	2F		.901	.000		.736	.241		.612	.020
	3F	.901	.897	-.004	.494	.734	.239	.593	.621	.028
	6EX		.900	-.001		.739	.245		.612	.020
stock	2F		.762	.037		.755	.260		.575	.051
	3F	.725	.767	.043	.496	.751	.255	.524	.578	.054
	6EX		.769	.044		.747	.252		.564	.040
hepatitis	2F		.869	.005		.755	.000		.803	-.016
	3F	.864	.868	.004	.755	.759	.004	.819	.826	.007
	6EX		.864	.000		.760	.005		.821	.002
physics	2F		.498	.000		.499	.000		.504	.000
	3F	.498	.506	.008	.499	.499	.000	.504	.495	-.008
	6EX		.506	.008		.499	.000		.504	.000
german	2F		.751	.011		.609	.118		.607	-.001
	3F	.740	.723	-.017	.492	.606	.115	.609	.593	-.015
	6EX		.736	-.004		.654	.162		.609	.000
crx	2F		.762	.000		.646	.000		.653	.014
	3F	.762	.779	.017	.646	.670	.024	.639	.673	.034
	6EX		.755	-.007		.722	.076		.667	.028
bands	2F		.779	.029		.611	.108		.559	-.096
	3F	.750	.747	-.003	.504	.603	.099	.655	.644	-.012
	6EX		.744	-.006		.580	.076		.655	.000
boa1	2F		.596	.024		.538	.041		.509	-.005
	3F	.571	.602	.031	.497	.548	.050	.515	.509	-.006
	6EX		.589	.018		.553	.056		.509	-.005

The results in Table 11 indicate that the combinatorial feature fusion method is effective and most effective for the decision tree learning method. The overall impact of the methods is shown in Table 12, which summarizes the results for each combinatorial fusion strategy and learning method, over the ten data sets. It displays the average absolute improvement in AUC as well as the win-lose-draw (W-L-D) record over the 10 data sets.

TABLE 12
SUMMARIZED AUC RESULTS FOR TEN DATA SETS

	Bayes		DT		1-NN	
Strategy	AUC	W-L-D	AUC	W-L-D	AUC	W-L-D
2-fusion	0.009	5-1-4	0.105	7-0-3	0.016	5-4-1
3-fusion	0.009	6-4-0	0.103	8-1-1	0.023	5-4-1
6-exhaustive	0.003	3-6-1	0.115	9-0-1	0.027	6-1-3

The results form both tables indicate that decision trees benefit most from combinatorial fusion, with the one-nearest neighbor learning method also showing substantial improvement. We believe that the decision tree algorithm improves the most because without combinatorial fusion it is incapable of learning combinations of numeric features, since decision trees only examine a single feature at a time.

The results do not demonstrate that any of the three combinatorial feature-fusion strategies is a clear winner over the other two. The 6-exhaustive strategy performs best for decision trees and one-nearest neighbor, but performs worst for naïve Bayes. The results for the 2-fusion and 3-fusion strategies are comparable even though the 3-fusion strategy generates more combinations. Our detailed results indicate that with the 3-fusion method some “3-fused” features make it to the final feature set, but apparently these are not essential for good performance. The fact that the 2-fusion strategy performs competitively indicates that most of the benefits that one can achieve with our combination operator can be achieved by combining only two features.

We generated Table 13 to determine if the combinatorial feature fusion method is more effective for the four most skewed data sets, where less than 10% of the data belongs to the minority class. These results, when compared to Table 13, show that the combinatorial fusion method yields substantially greater benefits when evaluated on the most highly unbalanced data sets, when the decision tree and one-nearest neighbor methods are used (the results for Bayes are much less convincing). Because of the limited number of datasets analyzed, these results cannot be considered conclusive, but nonetheless are quite suggestive.

TABLE 13
SUMMARIZED AUC RESULTS FOR FOUR SKEWED DATA SETS

	Bayes		DT		1-NN	
Strategy	AUC	W-L-D	AUC	W-L-D	AUC	W-L-D
2-fusion	0.004	1-1-2	0.195	4-0-0	0.065	4-0-0
3-fusion	0.012	2-2-0	0.185	3-1-0	0.059	3-0-1
6-exhaustive	0.006	1-3-0	0.194	4-0-0	0.062	4-0-0

It makes some sense that our method is most beneficial for highly unbalanced data sets. Given the performance measure described in Section 2, which is based on the correlation between the fused features and the minority-class examples, we expect to generate features that are useful for classifying minority-class examples. Furthermore, it is often quite difficult to identify “rare cases” in data and algorithms that look at multiple features in parallel are more likely to find the subtle classification rules that might otherwise get overlooked [18].

Although our primary interest is in improving classifier performance with respect to the area under the ROC curve, our method can be used to improve accuracy as well. We repeated a subset of our experiments using accuracy instead of AUC when determining whether adding a fused feature improves the performance with the required level of statistical confidence. Table 14 provides these results when using the 2-fusion strategy. We did not repeat these experiments for the other two strategies because AUC is our primary measure of interest and because the three strategies appear to perform similarly.

TABLE 14
SUMMARIZED AUC RESULTS FOR FOUR SKEWED DATA SETS

Dataset	Bayes			Decision Trees			1-NN		
	w/o	w	Diff	w/o	w	Diff	w/o	w	Diff
bio	98.8	98.8	0.0	99.4	99.4	0.0	99.2	99.2	0.0
letter-a	98.4	98.4	0.0	98.6	98.6	0.0	98.9	98.9	0.0
income	92.0	92.0	0.0	94.5	94.5	0.0	92.4	92.4	0.0
stock	80.4	80.4	0.0	90.3	90.3	0.0	86.3	86.3	0.0
hepatitis	84.0	84.0	0.0	86.2	80.7	-5.6	89.3	89.3	0.0
physics	68.9	75.3	6.5	75.1	75.2	0.1	62.6	75.0	12.4
german	73.2	73.2	0.0	69.5	73.0	3.5	68.1	71.6	3.5
crx	70.1	70.1	0.0	60.3	75.1	14.9	60.4	73.6	13.2
bands	67.0	67.0	0.0	61.4	61.4	0.0	65.3	65.3	0.0
boa1	55.0	57.0	2.0	51.0	56.9	6.0	52.6	57.5	5.0

The results in Table 14 indicate that our combinatorial fusion method is also effective for accuracy. While many of the data sets show no improvement, in ten cases there was an increase in accuracy, while in only one case was there a decrease in accuracy. In virtually every case where the accuracy remains the same, the combinatorial fusion strategy did not add any fused features. Similar to what we saw for AUC, the naïve Bayes method shows the least improvement.

5 Related Work

There has been a significant amount of work on feature mining/feature construction and in this section we mention some representative work. We organize the work in this area based on the operator used to combine the features. In our work, for example, numeric features are combined by mapping their feature values to ranks and then averaging the values of these ranks.

One approach is to assume that the features represent Boolean values and then use the standard logical operators to combine the features [2]. Other methods, such as the X-of-N method [20] differ in some ways but can be used to implement most logical operators. These logic-based methods require that all features first be mapped into Boolean values. This is not necessarily difficult but loses information and can lead to other problems. For example, in a decision tree repeatedly partitioning a numeric feature into binary values can lead to data fragmentation. In contrast our method reduces this problem by combining multiple numeric features.

Other methods are much more ambitious in the operators they implement. Some systems implement multiple mathematical operators, such as +, -, ×, and ÷, and relational operators such as ≤ and ≥ [1] [15]. Because these systems provide a rich set of operators, it is not feasible for them to try all possible combinations and thus they tend to employ complex heuristics. Thus our method has the advantage of simplicity. Again, a key difference is that our method combines ranks, whereas this other work combines the scores.

Feature selection [8], which involves determining which features are useful and should be kept for learning, is often mentioned in the same context as feature construction. Although we did not discuss feature selection in this paper, the techniques described in this paper have been used to implement feature selection and we hope to investigate this topic in the future.

6 Conclusion

This paper examined how a method from information fusion could be applied to feature construction from numerical features. The method was described in detail and three combinatorial fusion strategies were evaluated on ten data sets and three learning methods. The results were quite positive, especially for the data sets with the greatest class imbalance. When measuring AUC, the methods were of greatest benefit to the decision tree learning method, although it also substantially improved the 1-nearest neighbor method. Our results also indicate that our method can improve accuracy.

The work described in this paper can be extended in many ways. Our analysis would benefit from additional data sets—including several highly imbalanced data sets. It would also be interesting to evaluate additional combinatorial feature-fusion strategies, other than the three we evaluated in this paper. However, we suspect more complex fusion strategies will not yield substantial further improvements, so we do not view this as a critical limitation of our current work.

We also think that the basic algorithm can be extended in several ways. We plan on evaluating heuristic methods that would prune feature combinations that perform poorly. A heuristic method would enable us to evaluate more complex fusion schemes while potentially reducing the computation time. In this same vein, we also wish to consider simplifying the method for deciding whether to add a feature. Currently we use a validation set and only add a feature if the improvement in performance passes a statistical significance test. While there are benefits to this strategy, it also increases the computational requirements of the algorithm.

7 References

[1] E. Bloedorn and R. S. Michalski, "Data-driven constructive induction in AQ17-PRE: a method and experiments," in *Proc. of the 3rd International Conference on Tools*, 1991.

[2] A. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, December 1997, 97(1-2):245-271.

[3] A. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition, 30, 7*(July 1997), 1145-1159.

[4] P. K. Chan and S. J. Stolfo, "Toward scalable learning with non-uniform class and cost distributions: a case study in credit card fraud detection," in *Proc. of the Fourth International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 1998, 2001, 164-168.

[5] W. Cohen, R. Schapire and Y. Singer, "Learning to order things," *Journal of Artificial Intelligence Research*, 10 (1999), 243-270.

[6] P. Flach and N. Lavrac, "The role of feature construction in inductive rule learning," in *Proc. of the ICML 2000 Workshop on Attribute-Value and Relational Learning: crossing the boundaries*, 2000.

[7] J. W. Gryzmala-Busse, Z. Zheng, L. K. Goodwin and W. J. Gryzmala-Busse, "An approach to imbalanced data sets based on changing rule strength,", in *Learning from Imbalanced Data Sets: Papers from the AAAI Workshop*, AAAI Press, 2000.

[8] I. Guyon and A. Elisseef, "An introduction to variable and feature selection," *Journal of Machine Learning Research,* 3 (2003), 1157-1182.

[9] D. F. Hsu, Y. Chung and B. Kristal, "Combinatorial fusion analysis: methods and practices of combining multiple scoring systems," *Advanced Data Mining Technologies in Bioinformatics. Hershey, PA: Idea Group Publishing*; 2006, 32–62.

[10] D. F. Hsu and I. Taksa, "Comparing rank and score combination methods for data fusion in information retrieval," *Information Retrieval*, 8 (3), 2005, 449-480.

[11] C. Ma, D. Zhou and Y. Zhou, "Feature mining and integration for improving the prediction accuracy of translation initiation sites in eukaryotic mRNAs", in *5th International Conference on Grid and Cooperative Computing Workshops*, 2006, 349-356.

[12] Z. Markov and I. Russell, "An introduction to the WEKA data mining system," in *Proc. of the 11th SIGCSE Conference on Innovation and Technology in Computer Science Education*. 2006, 367–368.

[13] C. J. Matheus and L. A. Rendell, "Constructive induction on decision trees," in *Proc. of the 11th International Joint Conference on Artificial Intelligence*, 1989, 645-650.

[14] D. J. Newman, S. Hettich, C. L. Blake and C. J. Merz. *UCI repository of machine learning databases* [http://www.ics.usi.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science. 1998.

[15] F. Otero, M. Silva, A. Freitas, and J. Nievola, "Genetic programming for attribute construction in data mining," in *Proc. of 6th European Conference,* April 14-16, 2003.

[16] F. Provost, T. Fawcett and R. Kohavi, "The case against accuracy estimation for comparing classifiers," in *Proc. of the 15th International Conference on Machine Learning*, Moran Kaufmann, 1998, 445-453.

[17] S. Scott and S. Matwin, "Feature engineering for text classification," in *Proc. of the 16th International Conference on Machine Learning*, 1999, 379-388.

[18] G. M. Weiss, "Mining with rarity: a unifying framework," *SIGKDD Explorations*, *6, 1* (Dec. 2004), 7-19.

[19] G. M. Weiss and H. Hirsh, "Learning to predict rare events in event sequences," in *Proc. of the 4th International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 1998, 359-363.

[20] Z. J. Zheng, "Constructing X-of-N attributes for decision tree learning," *Machine Learning*, 40, 1 (2000), 35-75.