A Combinatorial Fusion Method for Feature Mining

Ye Tian Gary Weiss D. Frank Hsu Qiang Ma
Department of Computer and Information Science
Fordham University
441 East Fordham Road
Bronx, NY 10458
{ytian, gweiss, hsu, ma}@cis.fordham.edu

ABSTRACT
This paper demonstrates how methods borrowed from information fusion can improve the performance of a classifier by constructing ("fusing") new features that are combinations of existing numeric features. This work is an example of local pattern analysis and fusion because it identifies potentially useful patterns (i.e., feature combinations) from a single data source. In our work, we fuse features 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, although it seems to improve the performance of some learning methods more than others.

General Terms
Algorithms, Performance, Experimentation

Keywords
Feature construction, classification, class imbalance, information fusion, combinatorial fusion analysis

1. INTRODUCTION
The performance of a classification algorithm is highly dependent on the descriptions associated with each example. For this reason, practitioners typically spend a great deal to time making sure that these descriptions are accurate and capture the key aspects of the data. A good practitioner will choose the features used to describe the data very carefully. However, deciding which information to encode and how to encode it in a feature 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 [18], 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. This work can also be considered an example of local pattern analysis and fusion because we identify potentially useful patterns in the data (i.e., feature combinations) from a single data source.

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, although quite challenging, are quite common and are typical in domains such as medical diagnosis [7], fraud detection [4], and in predicting equipment failures [20]. 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 [19]. 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 (yet another term for feature construction).

We describe our combinatorial feature-fusion method in detail in Section 2 and then describe our experimental methodology in Section 3. Our experiments will evaluate our combinatorial feature-fusion strategy on ten data sets, using three learning methods (naïve Bayes, decision trees, and nearest-neighbor). The results from these experiments are described and analyzed in Section 4. We then discuss related work in Section 5. We finish by discussing our main conclusions and areas for future work in Section 6.

2. COMBINATORIAL FEATURE FUSION
This section describes the basic combinatorial feature-fusion method. We begin by providing some basic background information in Section 2.1. In Section 2.2 we describe our combinatorial feature-fusion algorithm.

2.1 Background
Our combinatorial feature-fusion method constructs new features by combining old features. In Section 2.1.1 we introduce some

MMIS '07, August 12, 2007, San Jose, CA, USA.

basic terminology and describe *how* features are combined, or fused. Then in Section 2.1.2 we discuss a variety of strategies for selecting the features to be fused. The algorithm described in Section 2.2 will then describe how the fusion strategy and fusion mechanism can be used to construct a set of features that will often improve classifier performance.

2.1.1 Terminology and Basic Steps

In this section we will use a simple example to help explain the relevant terminology and preliminary steps related to feature fusion. This example will also be used in Section 2.2 to help explain the feature-fusion algorithm. Please note that 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 to the classifier, unchanged.

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. Note that in the first two cases a higher score is desirable and in the last case a lower one is to be preferred.

Table 1 introduces our sample data set. This data set contains eight examples, labeled A-H, with five numeric features, F1-F5, and a binary class variable with values 0 or 1. 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 (i.e., feature value) with a rank, where a lower rank is better. We convert each score into a rank using a *rank function*. In this paper the rank function 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. In this case 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 F2 for records C, A, and F are 1, 2, and 3, respectively.

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 then we use 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 they are never used again. The rank values are used when combining features and when invoking the learning algorithm (i.e., the rank values are used as the feature values).

Table 2: Sample data set 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

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 our 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. How many records do we look at? The number of records 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, the number of class "1" values in the top 3 records (recall that "1" 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. For this simple example, F1–F4 all have performances of 2/3 and F5 has a performance of 0. Table 4 shows the performances of each of the original "unfused" features. In this overly simplified example, four of the features all have the same performance.

Table 4: Performance values for original features

Feature	Perform-ance
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 combined (i.e., fused) features. However, to do this we need to determine the rank of a fused feature, so we can sort the examples by this rank. We compute this using a *rank combination function*. Our rank combination function 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

2.1.2 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. However, because this is generally not feasible, due to the number of features that would be introduced, we consider some more restrictive strategies. 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 would therefore equal $C(n,1) + C(n,2) + \ldots C(n, n)$, 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 numeric features ($k < n$). For example, a 6-exhaustive strategy for a data set with 20 numeric features will select 6 numeric features and then fuse them in all possible ways. Doing so will reduce 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 4. Because it will not be expensive to include all of the original features, we also include the $n - k$ original features not used in the fusion process. 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 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 (there are $C(n,1)$ of these). Note that 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 since we would include the three original features which were excluded).

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**

2.2 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 a complete example, based on the data set introduced in Table 1 of Section 2.1.

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 in Table 7 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. As mentioned previously, the data and features are from Table 1. The next few steps were already described in Section 2.1.1. First we convert the scores to ranks (line 3). We then calculate the performance of each 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 its performance. This is essentially the same steps that were done earlier for the original, unfused, features. We then sort the features by decreasing performance value (line 17) and then extract the features from this sorted list and save them (line 18) in Candidates, the ordered list of candidate fused features. The results for our simple example are shown in Table 8. We show only the 14 best performing fused features. In this case Candidates would equal {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

We have now completed the first half of the algorithm. In the second half, 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 (although slightly more difficult to describe). In the actual implementation 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. 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.

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 provide the background necessary to understand our experiments. In Section 3.1 we describe the datasets employed in our empirical study and in Section 3.2 we describe the three learning methods that we utilize. Section 3.3 then describes our experimental methodology.

3.1 Datasets

The ten data sets used in our study are described in Table 10. The first field provides the data set name, the second the percentage of examples belonging to the minority class, the third specifies the final number of features and the last column lists the data set size. The data sets are ordered in terms of decreasing class imbalance.

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

The data sets come from a few sources. The hepatitis, bands, income and letter-a data sets were obtained from the UCI machine learning repository [14] and the crx data set was provided in the Data directory that came with the C4.5 code. The boa1 data set was obtained from researchers at AT&T and has been used in previous published work. 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.

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 "+".

3.2. Learning Methods

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.

3.3. Experimental Methodology

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 still 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 3.2 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 (2-F), 3-fusion (3-F) and 6-Exhaustive (6-EX). 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

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

The results in Table 11 indicate that the combinatorial feature fusion method is effective. The results appear to be most positive for the decision tree learning method. The overall impact of the methods is shown in Table 12, which summarizes the results of the ten data sets. Table 12 shows the summarized results for each combinatorial fusion strategy and learning method. 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 Set

	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 showing the second-best improvement. Because the maximum AUC value is 1.0, even an average improvement of .023 (for 1-NN using the 3-fusion strategy) is quite substantial. We believe that the decision tree algorithm improves the most because it is incapable of learning combinations of numeric features, because it can only examine a single feature at a time (oblique decision trees are not subject to this limitation). That is, decision trees operate by making axis-parallel cuts in the "concept space" and because of this limitation, it cannot correctly learn a simple concept such as $x + y = 1$ (although it can approximate it). However, with our combinatorial feature fusion method, we would expect a traditional decision tree to easily learn such a concept.

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 does perform the best for decision trees and one-nearest neighbor, but performs the worst for naïve Bayes. Since the 3-fusion strategy subsumes the 2-fusion strategy (i.e., it generates a superset of the features generated by the 2-fusion strategy) it is worthwhile to compare these two strategies. Because the results are quite similar, the best we can say is that the results are comparable. Our individual results do show that with the 3-fusion method some 3-fused features are generated and used in the final feature set, so we can conclude that the strategies do differ in what features are used (the same is true of the 6-exhaustive strategy). The fact that the 2-fusion strategy performs competitively with the others may indicate that in practice most of the potential benefits that one can achieve with our combination operator can be achieved by combining only two features.

We can analyze the results in Table 11 to determine if the combinatorial feature fusion method is most effective for data sets with the highest levels of class imbalance. The first four data sets listed in the table all have less than 10% of the data set belonging to the minority class (i.e., have a class ratio greater than 9:1). Table 13 shows the average AUC performance over just these four data sets.

Table 13: Summarized AUC Results for 4 Skewed Datasets

	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

The results in Table 13, when compared to Table 12, show that the combinatorial fusion method is substantially more beneficial, when using the decision trees and one-nearest neighbor method, for the most skewed data sets (i.e., the ones with the most class imbalance). Between these two methods, the improvement averages about twice the improvement over the ten data sets. The Bayes method shows an improvement in AUC also, but given the win-loss-tie record this is less convincing. Because of the limited number of datasets analyzed, these results cannot be considered conclusive, but nonetheless are quite suggestive. There are two explanations for the more substantial improvement for the most highly skewed data sets. First, because the performance measure described in Section 2 is based on the correlation between the fused feature and the minority-class examples, the features that are

more likely to improve minority-class performance are considered first. This makes them more likely to be added, since adding other features first might obscure these improvements. Secondly, it is often quite difficult to identify "rare cases" and algorithms that look at multiple features in parallel are more likely to find the subtle classification rules that might otherwise get overlooked [19].

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 to determine whether adding a fused feature improves the performance of the classifier with the required level of statistical confidence. Table 14 provides the 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: Accuracy Results using 2-Fusion Strategy

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 most of the data sets show no improvement, only in one case did the combinatorial fusion strategy lead to a decrease in accuracy. In contrast, in ten cases there was an increase 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 so in this section we only mention some representative work. One way to organize work in this area is by the operator used to combine the features. In our work, for example, numeric features are combined by mapping their feature values (i.e., scores) 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 (e.g., $\wedge$,$\vee$) to combine the features[2]. Other methods, such as the X-of-N method [21] differ in some ways but can nonetheless be used to implement most of the logical operators. These logic-based methods require that categorical features and numerical features first be mapped into Boolean values. This is not necessarily difficult, since data reduction algorithms already exist for this (e.g., C4.5 [17] can generate binary splits for numerical features), but this step loses information and, especially for numerical features, may not be a natural choice. This mapping can also lead to other problems. For example, with decision trees, repeatedly partitioning a numeric feature into binary values can lead to data fragmentation, whereas our method actually reduces this problem by allowing one to combine multiple numeric features directly.

Other methods are much more ambitious in the operators they implement. For example, some systems implement multiple mathematical operators, such as "+", "-", "×", and "÷", and relational operators such as "≤" and "≥" [1, 15]. Because these systems provide a richer set of operators than we do, it is not feasible to try all possible combinations, such as our exhaustive fusion strategies. These methods rely on more complex, search-based, heuristic methods. 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.

It is informative to describe our work using a general framework for feature construction that has been proposed [13]. This framework involves the following four steps: 1) detection of when feature construction is required, 2) selection of constructors, 3) generalization of the selected constructors, and 4) evaluation of the new features. The first step involves detecting when there is a need for feature construction. We use a "no detection" policy since we always perform feature construction. Other options would be to only construct features when the classifier that is initially produced fails to meet a pre-specified requirement (e.g., accuracy is too low). The next step involves selecting the constructors to use, which in our case is simple since we only have a single constructor (i.e., combination operator). We do not generalize during the feature construction process, so our work is *not* an example of constructive induction. Finally, we evaluate our new features by tentatively adding them to the classifier one at a time, based on their ranked performance, but only keep the feature if we are statistically confident that it will improve classifier performance.

Feature selection [8], which involves determining which features are useful and should be selected, is often mentioned in the same context as feature construction. Although we did not discuss feature selection in this paper, we have used the techniques in this paper to implement feature selection and plan to investigate this topic in the future. In particular, the measure of feature performance that is introduced in this paper can be used to select the most useful features and to prune feature with low performance.

6. CONCLUSION

This paper examined how a method from information fusion could be applied to feature construction. This method was described in detail and then three combinatorial fusion strategies were evaluated on ten data sets and three learning methods. This combinatorial feature-fusion method was applied to numeric features and our 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 performance of the 1-nearest neighbor method. Our results also indicate that this method is effective at improving accuracy.

The work described in this can be extended in many ways. While the ten data sets we analyzed is a reasonable number of data sets, it clearly would be a benefit to evaluate our methods on additional data sets, including several additional, 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 our basic algorithm can be extended in a few ways. First, although we rank our features and feature combinations and then evaluate their performance based on their correlation with minority-class examples, these are only used to determine the order in which the fused features are considered for inclusion. We plan on evaluating heuristic methods, which would prune feature combinations that perform poorly. This would enable us to evaluate more complex fusion schemes with similar effort or to improve the computational performance of the algorithm. 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.

Finally, we would like to apply our combinatorial method to numeric features using operators that do more than *average* the ranks of the features. While this would greatly expand the space of possible combinations, it could lead to further improvements.

7. REFERENCES

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

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

[3] Bradley, A. 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] Chan, P.K., and Stolfo, S.J. Toward scalable learning with non-uniform class and cost distributions: a case study in credit card fraud detection. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (*KDD '98)*, (New York, NY, Aug. 27-31, 1998), AAAI Press, 1998, 2001, 164-168.

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

[6] Flach, P. and Lavrac, N. The role of feature construction in inductive rule learning. In: *Proceedings of the ICML2000 workshop on Attribute-Value and Relational Learning: crossing the boundaries*, 2000.

[7] Gryzmala-Busse, J. W., Zheng, Z., Goodwin, L. K., and Gryzmala-Busse, W. J. An approach to imbalanced data sets based on changing rule strength. In *Learning from Imbalanced Data Sets: Papers from the AAAI Workshop* (Austin, TX, July 31, 2000), AAAI Press, 2000.

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

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

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

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

[13] Matheus, C.J. and Rendell, L.A. Constructive induction on decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989, 645-650.

[14] Newman, D. J., Hettich, S., Blake, C.L., and Merz, C. J. *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] Otero, F., Silva, M., Freitas, A. and Nievola, J. Genetic programming for attribute construction in data mining. In *Proceedings of 6th European Conference,* April 14-16, 2003.

[16] Provost, F., Fawcett, T., and Kohavi, R. The case against accuracy estimation for comparing classifiers. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)* (Madison, Wisconsin, July 24-27, 1998), Morgan Kaufmann, 1998, 445-453.

[17] Quinlan, J. R. C4.5: Programs for machine learning. Morgann Kaufmann, San Mateo, CA, 1993.

[18] Scott, S. and Matwin, S., Feature engineering for text classification. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999, 379-388.

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

[20] Weiss, G. M., and Hirsh, H. Learning to predict rare events in event sequences. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (*KDD '98)*, (New York, NY, Aug. 27-31, 1998), AAAI Press, 1998, 359-363.

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