

Generalized and Heuristic-Free Feature Construction for Improved Accuracy

Wei Fan* Erheng Zhong† Jing Peng‡ Olivier Verscheure* Kun Zhang§ Jiangtao Ren† Rong Yan* Qiang Yang¶

Abstract

State-of-the-art learning algorithms accept data in feature vector format as input. Examples belonging to different classes may not always be easy to separate in the original feature space. One may ask: can transformation of existing features into new space reveal significant discriminative information not obvious in the original space? Since there can be infinite number of ways to extend features, it is impractical to first enumerate and then perform feature selection. Second, evaluation of discriminative power on the complete dataset is not always optimal. This is because features highly discriminative on subset of examples may not necessarily be significant when evaluated on the entire dataset. Third, feature construction ought to be automated and general, such that, it doesn't require domain knowledge and its improved accuracy maintains over a large number of classification algorithms. In this paper, we propose a framework to address these problems through the following steps: (1) divide-conquer to avoid exhaustive enumeration; (2) local feature construction and evaluation within subspaces of examples where local error is still high and constructed features thus far still do not predict well; (3) weighting rules based search that is domain knowledge free and has provable performance guarantee. Empirical studies indicate that significant improvement (as much as 9% in accuracy and 28% in AUC) is achieved using the newly constructed features over a variety of inductive learners evaluated against a number of balanced, skewed and high-dimensional datasets. Software and datasets are available from the authors.

Keywords: Feature Construction, Accuracy Improvement, Automatic, Efficiency

1 Introduction

For most inductive learning algorithms, examples are assumed to be in feature vector format. Having a good set of features is the key to high accuracy. There are various reasons why a given feature set may have only limited discriminative power. For example, each feature itself has little information gain, but some significant discriminative information can be revealed if a number of these features are combined and transformed. For problems like this, feature selection is often not very effective, as its basic assumption that good features exist in the original feature set is violated. As a result, techniques that consider multiple features at the same time or their nonlinear transformation [10] become important. Let us consider a synthetic example in Figure 1. Figure 1(a) shows an XOR-like problem, whereas positive and negative examples cannot be separated by any linear decision boundaries. However, if we construct a new feature $F_3 = F_1 * F_2$ and project the data points onto the space spanned by F_1 and F_3 instead (Figure 1(b)), a simple linear boundary can perfectly separate the two classes. This example shows that constructing new features can be very useful to capture intricate complexity in the instance space, thus to closely represent and model the concept of interest. As compared to feature selection, there is a relatively fewer methods on feature construction. As reviewed below, existing methods have problems of inefficiency, non-optimality and use of domain heuristics.

1.1 Limitations of Existing Approaches Some techniques [14, 17] utilize a two-step batch-process that first builds a set of expanded features, and then perform feature selection. Since the combinatorics to construct new features from original features can be infinite, it is impossible to exhaust all candidates and select the most discriminant ones. Even we limit operators to come from a finite set, these methods can still produce a significantly large number of new features that are inefficient in practice. For example,

*IBM T. J. Watson Research Center, USA. {weifan, ov1, yanr}@us.ibm.com.

†Sun Yat-Sen University, Guangzhou, China. {sw04zheh@mail2, issrjt@mail}.sysu.edu.cn.

‡Department of Computer Science, Montclair State University, USA. pengj@mail.montclair.edu.

§Department of Computer Science, Xavier University of Louisiana, USA. kzhang@xula.edu.

¶Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong. qyang@cse.ust.hk.

if there are only 4 binary operators and the number of original features is n_f , there can be up to $4 \times n_f^2$ new features. On the other hand, kernel-based methods, e.g., kernel discriminant analysis (KDA) [1] and kernel principal component analysis (KPCA) [16], compute good features without exhaustive search. For KDA, $n_c - 1$ most discriminant features over the entire data set are chosen for classification, where n_c represents the number of classes. However, this might not be always optimal. A feature insignificant on the whole data set can be highly discriminant in a local region that is not covered well by features constructed so far. Let us consider the example in Figure 2. F_1 is obviously the most discriminant over the entire data set, and can be the only feature chosen by a feature selection technique. Thus, F_2 can be overlooked, but is actually highly predictive in the region above the horizontal dashed line. Some methods employ decision tree to avoid the above two problems [11, 13, 15, 12], but they require domain knowledge. The approach in [12] requires such knowledge to select constructors as well as filters to reject undesirable candidate features. Similarly, the framework described in [11] uses specifications written by the user, based on knowledge of the application.

1.2 The Proposed Approach To address these challenges, we propose a generalized approach based on local feature construction, divide-conquer search and weighting rule-based operator selection. The basic idea is to build a decision tree by evaluating and selecting both the original and newly computed features. At each node, an operator most likely to produce good features is chosen to build new features. The decision as to which feature (either new or original) to choose and then split the node is evaluated on the local data or subset of examples contained in the current node. Candidates include information gain, gain ratio, gini-index and Kearns-Mansour criteria. The recursion stops when either examples in the node all belong to the same class or the number of examples is below a given threshold. The main flow is summarized in Figure 3(a). Once a tree has been built, features chosen at internal decision nodes are the set of constructed features to project the data. In addition, the tree itself is a classifier. During the feature construction and evaluation process, an adjustment rule updates weights associated with each operator, to give more weights to those having produced highly predictive features in earlier steps. Thus, these good operators have more chance to be chosen again. The detail is illustrated in Figure 3(b).

To summarize, the proposed method constructs features on the basis of finite but dynamically chosen and adjusted set, thereby to avoid searching in a large

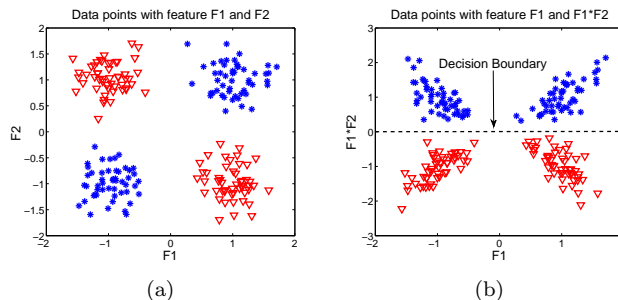


Figure 1: Synthetic Example of Feature Construction. The $\nabla/*$ refers to positive/negative.

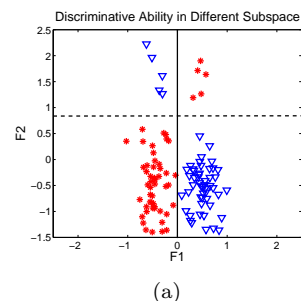


Figure 2: Different Discriminability. The $\nabla/*$ refers to positive/negative.

and potentially infinite pool. It estimates features' significance on local data subspaces rather than always on the complete data set. This insures optimal coverage for the set of constructed features as a whole on the entire dataset. In addition, the weights of constructor operators are dynamically adjusted according to their performance on the dataset, therefore, avoid the use of domain knowledge. As stated in Section 3, this approach avoids over-fitting with performance bound guarantee.

2 Local Feature Construction

In this section, we formulate the local feature construction problem, followed by a discussion on the divide-and-conquer based solution. The notations and terminologies are summarized in Table 1.

2.1 Problem Formulation Let X and Y be the instance and label space. Suppose that the training data $L = \{X_L, Y_L\}$ contain ℓ instances, $X_L = \{\mathbf{x}_1, \dots, \mathbf{x}_\ell\}$, $Y_L = \{y_1, \dots, y_\ell\}$. Assume the original feature set of the data is \mathcal{F} with n_f continuous features, and the number of class labels is n_c . This paper focuses on the binary classification problem, namely $n_c = 2$ and $Y = \{-1, 1\}$. Also, let Δ be the set of operators or

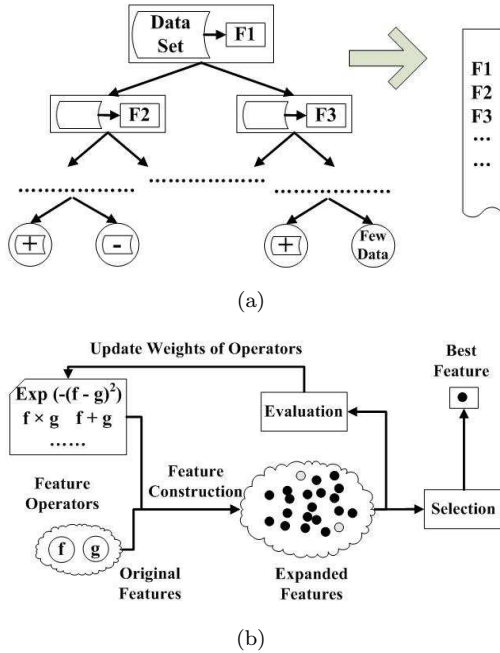


Figure 3: Main Flow of Divide-conquer based Feature Construction

mapping functions that can be applied to the original features \mathcal{F} . Because of the unlimited number of ways to apply Δ , the extended feature set \mathcal{A} can be infinite. The problem of feature construction is defined as follows,

DEFINITION 2.1. (Feature Construction) A constructed feature $\alpha_i = op_k(\mathcal{F}_s)$ is an element in \mathcal{A} , where op_k is the k -th operator in Δ , and \mathcal{F}_s is a subset of \mathcal{F} . Furthermore, we define the constructed feature set as $\mathcal{E} = \{\alpha_1, \dots, \alpha_{n_t} \in \mathcal{A}\}$, where n_t is the number of constructed features.

After feature construction, both original feature set \mathcal{F} and constructed feature set \mathcal{E} can be used for classification. There are three aspects inherent to the problem of feature construction:

1. Search the feature set \mathcal{F} for the best features.
2. Select the constructors Δ .
3. Evaluate the constructed features.

2.2 Algorithm Algorithm 2 shows the main flow of the divide-and-conquer based feature construction approach, called FCTree (Feature Construction Tree). The basic idea is to partition the training data top-down using the best features, which can be either the original or those automatically constructed. At each node, we

Table 1: Definition of notation

Notation	Notation Description
X	Instance space
Y	Label space
L	Training data
ℓ	Number of examples in L
Δ	Base operators set
\mathcal{F}	Original feature set
\mathcal{E}	Constructed feature set
\mathcal{A}	Feature set can be extended from \mathcal{F}
\mathcal{W}_O	Weights of operators
n_f	Number of original features
n_c	Number of classes
n_e	Number of expanded features in each node
n_t	Number of features in \mathcal{E}
n_o	Number of operators
$I_f(f)$	Information gain of feature f

Algorithm 1 Update Weights

- 1: **Input:** Constructed features at the current node: \hat{F} , Weights of operators: \mathcal{W}_O
- 2: **Output:** New weights of operators: \mathcal{W}_O
- 3: Calculate the information gain of each feature in \hat{F}
- 4: Calculate the criterion value of each operators I , using Eq.(2.1)
- 5: Update \mathcal{W}_O : $\forall w_j \in \mathcal{W}_O, w_j = w_j * \exp(-\frac{1}{1+I_j})$
- 6: Normalize \mathcal{W}_O : $\forall w_j \in \mathcal{W}_O, w_j = w_j / \sum_{w_j \in \mathcal{W}_O}$
- 7: **Return** \mathcal{W}_O

generate a new feature set \hat{F} as follows: we randomly select an operator from Δ according to their weights and a subset of features from \mathcal{F} to construct a new feature α . This process is iterated for n_e times. Then, the best feature is selected from \hat{F} and \mathcal{F} according to information gain computed from the examples at the current node. We partition examples at the current node into disjoint subsets by testing against the selected feature. Then, the selected feature is removed from the feature pool to avoid being chosen again. After that, the weights associated with the operators are updated. The idea here is to allocate more weights to those operators having produced highly discriminate features in earlier steps, so that they are more likely to be selected for feature construction at later iterations. The weights are updated as follows. Let I be the criterion values of Δ , I_k be its k -th element, n_k be the number of features in \hat{F} generated by the k -th operator and I_{ik}^f be the information gain of i -th feature constructed by the k -th

Algorithm 2 FCtree

- 1: **Input:** Original features: \mathcal{F} , Training data: L , Operators: Δ , Number of constructed features at each node n_e
 - 2: **Output:** Constructed features: \mathcal{E} , Decision Tree: $FCTree$
 - 3: Let every element in \mathcal{W}_O be $1/\ell$, $\mathcal{E} = \emptyset$
 - 4: Let root node of the tree as $\text{Node}(L, \mathcal{W}_O)$
 - 5: For the current node, generate a subset $\hat{F} = \emptyset$
 - 6: **for** $j=1$ to n_e **do**
 - 7: Select an operator op from Δ according to \mathcal{W}_O by weighted roulette wheel
 - 8: Select features \mathcal{F}_t from \mathcal{F} randomly
 - 9: Generate a new feature $\alpha_j = op(\mathcal{F}_t)$
 - 10: $\hat{F} = \hat{F} \cup \alpha_j$
 - 11: **end for**
 - 12: Combine \hat{F} and \mathcal{F} , $S_f = \hat{F} \cup \mathcal{F}$
 - 13: Select the best discriminative and not used before feature f^* from S_f
 - 14: **IF** $f^* \in \hat{F}$ **Then** Add f^* into \mathcal{E} , $\mathcal{E} = \mathcal{E} \cup f^*$
 - 15: Update \mathcal{W}_O using Algorithm 1
 - 16: Split node into two sub-nodes based on f^* , obtaining $\text{Node}(L_l, \mathcal{W}_O)$ and $\text{Node}(L_r, \mathcal{W}_O)$
 - 17: Repeat the above process for the sub-nodes until the node is “pure” or just contains very few instances
 - 18: Let $FCTree$ be the obtained decision tree
 - 19: **Return** \mathcal{E} , $FCTree$
-

operator. Then, I_k is defined as:

$$(2.1) \quad I_k = \sum_{i=1}^{n_k} \frac{I_{ik}^f}{n_k}$$

The weights of Δ are updated using the following rule:

$$(2.2) \quad w_k = w_k * \exp\left(-\frac{1}{1 + I_k}\right)$$

where w_k is the weight of the k^{th} operator in Δ and it is normalized at each iteration.

The FCtree process recursively partitions the data and builds best features until the following stopping conditions are met: (1) the number of instances in the node is smaller than a threshold, e.g. $m = 3$; (2) the node only contains examples from one class. At the end, we combine all features from \mathcal{E} and \mathcal{F} to form a new feature space. At the same time, FCtree also generates a decision tree classifier that can be used for prediction.

3 Formal Analysis

We analyze the following problems related to the three aspects stated in Section 2.1: (1) How does the proposed feature construction method avoid exhaustive search

and obtain good stability? (2) What role does the weighting rule play in FCtree? (3) How discriminant are the features selected by FCtree, including the original and constructed ones?

3.1 Scalability of FCtree The first concern is how many features will be constructed and returned in the proposed method. Here we provide a bound on search space. Let ℓ be the number of training examples. As stated in Section 2.2, the number of instances in one leaf node cannot be less than a threshold m . When all leaf nodes in a FCtree contain m instances, its size is maximal in terms of the number of leaf nodes. As a result, the number of constructed features N_t has the following upper bound

$$(3.3) \quad N_t = O((2^{\log_m \ell} - 1) * n_e) < O(\ell * n_e),$$

where n_e is the number of constructed features at each node. Thus, the number of selected constructed features n_t will not exceed $O(\ell)$ because FCtree selects only one feature at a node. From Eq.(3.3), we observe that the number of constructed features is bounded and will not exceed the number of training examples.

From the above bound, we analyze the complexity of the FCtree algorithm. In the worst case, it requires $O(\log_2 \ell * \ell * n_e)$ computations to evaluate and select the best features and $O(\ell * n_o)$ computations to update the weights associated with operators, where n_o is the number of operators. In addition, it takes $O(\log_2 \ell * \ell)$ computations for all instances to traverse down the tree (from the root to a leaf node). Thus, given that n_o is substantially smaller than $n_e = O(n_f)$, and in practice the proposed FCtree algorithm has the complexity of $O(\log_2 \ell * \ell * n_f)$, where n_f represents the number of original features.

For comparison, let us consider the computational complexity of batch processing. Suppose that the operators are all binary. Batch processing will need $O(n_o * n_f^2)$ computations to generate features and $O(n_o * n_f^2 * \ell)$ computations to evaluate them. This implies that the computational complexity of batch processing is given by $O(n_o * n_f^2 * \ell)$. That is, the complexity of batch processing grows quadratically with the number of features. In contrast, the complexity of FCtree grows linearly with n_f . Thus, it is much more efficient in practice.

3.2 Analysis of the Weighting Rule We show that if an operator has a higher weight at the current node, it is expected to perform well in its splits.

As defined in Section 2.2, operators that produce features with higher information gain will receive higher weight. Thus, suppose that at the current node we have

two operators op_1 and op_2 with $w_1 > w_2$. Then

$$E_{f_1 \sim op_1}[I^f(f_1)] > E_{f_2 \sim op_2}[I^f(f_2)]$$

where $E[*]$ is the expectation value of $*$, f_1 is the feature produced by op_1 , f_2 is produced by op_2 , and $I^f(f)$ is the information gain of feature f . Assume that when splitting the data along f_1 and f_2 , both place most positive instances at the right sub-node and most negative instances at the left sub-node. As follows, we show that if $I^f(f_1) > I^f(f_2)$ at the current node, then on average this inequality still holds at the two sub-nodes.

Assume that the current node is split by feature f^* . There are P_l positive instances and N_l negative instances at the left sub-node, P_r positive and N_r negative instances at the right sub-node. In addition, we use P_{l1} , N_{l1} , P_{r1} and N_{r1} to represent the split result if f_1 is selected. Similarly, we use P_{l2} , N_{l2} , P_{r2} and N_{r2} to indicate the result when using f_2 . For ease of discussion, we assume $P_{l1} > P_{r1}$, $N_{l1} < N_{r1}$, $P_{l2} > P_{r2}$ and $N_{l2} < N_{r2}$ because $I^f(f_1) > I^f(f_2)$, $P_{l1} > P_{l2}$ and $N_{l1} < N_{l2}$. Now, we consider the left sub-node. The analysis of the right sub-node is similar. On average, there are $P_{l1} \times \frac{P_l}{P_l+P_r} + P_{r1} \times \frac{P_l}{P_l+P_r}$ positive instances and $N_{l1} \times \frac{N_l}{N_l+N_r} + N_{r1} \times \frac{N_l}{N_l+N_r}$ negatives in the left sub-node for feature f_1 . Also, for feature f_2 , there are $P_{l2} \times \frac{P_l}{P_l+P_r} + P_{r2} \times \frac{P_l}{P_l+P_r}$ and $N_{l2} \times \frac{N_l}{N_l+N_r} + N_{r2} \times \frac{N_l}{N_l+N_r}$ instances, respectively. Moreover, we define

$$\begin{aligned} P'_{l1} &= P_{l1} \times \frac{P_l}{P_l + P_r} \\ N'_{l1} &= N_{l1} \times \frac{N_l}{N_l + N_r} \\ P'_{r1} &= P_{r1} \times \frac{P_l}{P_l + P_r} \\ N'_{r1} &= N_{r1} \times \frac{N_l}{N_l + N_r} \end{aligned}$$

In addition, f_2 has similar relationships. Because $P_{l1} > P_{l2}$ and $N_{l1} < N_{l2}$, we have $P'_{l1} > P'_{l2}$ and $N'_{l1} < N'_{l2}$. This means the two sub-nodes produced by f_1 are “purer” than those by f_2 . Thus, the information gain of f_1 at the left sub-node is expected to be higher than the one obtained by f_2 . We can conclude that the operator good at the current node is expected to be good at two sub-nodes.

3.3 Feature selection under exhaustive search

If we were able to enumerate all features in \mathcal{A} a priori, we could consider FCtree as a feature selection algorithm. We show that under this ideal situation, the features selected by FCtree are still the best ones. We compare FCtree with a forward-based feature selection al-

gorithm using a decision tree, called fDT as in [4]. Without loss of generality, we assume $\mathcal{A} = \{\alpha_1, \dots, \alpha_{n_a}\}$ is finite. Furthermore, we assume that both fDT and FCtree select K features from the candidates in \mathcal{A} and FCtree obtains K features when it reaches the stopping condition.

Once fDT has selected k features, it chooses the $(k+1)$ -th feature from remaining ones to achieve the highest prediction accuracy measured using the k selected features. Without loss of generality, we assume that the accuracy never decreases before it obtains K features. On the other hand, at each iteration, FCtree selects a feature from the original and constructed features. Importantly, this feature has not yet been used along the decision path starting from the root to the current node, and achieves the maximal accuracy increase for those instances at the current node. By simple induction, the analysis below shows that fDT and FCtree choose the same subset of features. The analysis is adopted from [4].

At the root node, both FCtree and fDT will select the same feature because they evaluate features using the same criterion on the same data set. Assume after selecting $k-1$ features, fDT and FCtree construct the same partial tree. We show that when considering the k -th feature, the trees built by fDT and FCtree will be the same. First, neither fDT nor FCtree will reconstruct the partial tree. For fDT, if a new feature would reconstruct this partial tree, it would have been chosen previously and already been the selected feature of a non-terminal node of the partial tree. For FCtree, it does not select those features along the path from the root to the current node. Thus, FCtree does not reconstruct the partial tree. Therefore, when fDT and FCtree expand identical nodes, there is only one unique best feature to select for both FCtree and fDT.

3.4 Error Bound for FCtree classifier

One important advantage to use a decision tree for feature construction is that the decision regarding which feature to select for splitting is entirely based on the local data contained in each node (or region). Thus, it potentially creates features that provide “local” relevance at different prediction locations in the feature space. When they are combined, they provide a good coverage for the feature space. We now provide an error bound for the FCtree algorithm, as adapted from [6]. For a tree with N leaves, the leaf function h_i , the conjunction of all tests on the path from the root to leaf i , can be defined as $h_i : X \rightarrow \{0, 1\}$ by $h_i(\mathbf{x}) = 1$ iff \mathbf{x} reaches leaf i , for $i = \{1, \dots, N\}$.

Then for every internal decision node, we choose one feature from original and constructed features to split

the node into two sub-nodes. Notice that constructed features are simply a function of the original features. Threshold functions (original and constructed features) form a class \mathcal{G} of boolean functions that label internal decision nodes. For the labeled data L , let $Q_i = P_\ell[h_i(\mathbf{x}) = 1]$. Then $Q = (Q_1, \dots, Q_N)$ is a probability vector. Let us define the class of leaf functions for leaves up to depth j as

$$(3.4) \quad \mathcal{H}_j = \{h : h = g_1 \wedge \dots \wedge g_r | r \leq j, g_i \in \mathcal{G}\}$$

Then the VC dimension of \mathcal{H}_j , denoted by $d_{vc}(\mathcal{H}_j)$, is less than or equal to $2j * d_{vc}(\mathcal{G}) * \ln(2e_j)$. Let d_i denote the depth of leaf i , so $h_i \in \mathcal{H}_{d_i}$, and let $d = \max_i d_i$. Moreover, let $\rho(Q, U_p) = \sum_{i=1}^N (Q_i - 1/N)^2$ be the quadratic distance between the probability vector P and the uniform probability vector $U_p = \{1/N, \dots, 1/N\}$. Thus, the effective number of leaves in the tree can be defined as $N^* = N(1 - \rho(Q, U_p))$. From these definitions, the error bound for FCTree in terms of N^* can be established based on the labeled data.

THEOREM 3.1. *For a fixed $\theta > 0$, there is a constant c satisfies the following. Let \mathcal{D} be the distribution on $X \times Y$. Consider the decision tree T achieved by FCTree with depth d and decision function in \mathcal{G} . With probability at least $1 - \theta$ over the training set L , T is consist with L has*

$$(3.5) \quad P_{\mathcal{D}}(T(\mathbf{x}) \neq y) \leq c \left(\frac{N^* d_{vc}(\mathcal{G}) \log^2 \ell \log d}{\ell} \right)^{1/2}$$

where N^* is the effective number of leaves of T .

The detail of the proof can be found in [6].

Notice that N^* and $d_{vc}(\mathcal{G})$ are the two dominant terms in the bound. While a tree created by FCTree potentially has a higher complexity class \mathcal{G} of boolean functions due to constructed features, the same tree can have a much smaller effective number of leaves, again due to constructed features. This can be seen from the simple example shown in Figure 1, where a tree with constructed features has two leaves, while a tree with the original features only that is consistent with the sample points requires four leaves. If a decrease in the effective number of leaves outweighs an increase in the complexity of the function class \mathcal{G} , a tighter bound can be achieved. This shows that FCTree has the potential to achieve better generalization performance, as we shall see later.

4 Experimental Results

Twenty two real-world datasets collected from four different domains are used to empirically evaluate the proposed algorithm. The performances measured in

Table 2: Description of Datasets

Dataset	n_f	Size	Dataset	n_f	Size
UCI			Landmine		
Sonar	60	208	L1	9	690
Ionosphere	34	351	L2	9	690
Masses	5	961	L3	9	689
Transfusion	4	748	L4	9	508
Parkinson	22	195	L5	9	509
Caltech-256			Nuclear		
Caltech1	177	197	V	4	1703
Caltech2	177	249	W	4	777
Caltech3	177	225	X	4	2300
Caltech4	177	248	Y	4	1209
Caltech5	177	224	Z	4	2705
Caltech6	177	278	All	4	8695

accuracy or AUC are compared with standard machine learning algorithms on different feature spaces. Three sets of studies are also conducted to further examine the sensitivity and scalability of the proposed method, as well as the strength of the weighting rule. In addition, we illustrate the utility of feature construction using an image categorization example.

4.1 Dataset Description and Experimental Setting

Table 2 summarizes the statistics of twenty two binary datasets respectively obtained from UCI repository¹, Caltech-256 database [8], Landmine collection² and Nuclear Ban data source³. Among them, Landmine contains data collected from real landmines via remote sensing techniques. Each data point is represented by a 9-dimensional feature vector extracted from radar images, and the class label is either true or false mine. Caltech-256 is an image database of 256 object categories. To form the six binary classification problems listed in this group, we select four categories and they are “ak47”, “American-flag”, “backpack” and “baseball-bat”. Each category is processed via a 177-dimensional color correlogram [9]. Nuclear Ban data source is a nuclear explosion detection problem used by ICDM’08 contest. “ $V \sim Z$ ” denote the data collected at each individual station, and “All” refers to the entire dataset from all stations. In addition, relatively balanced class distribution is preserved in the datasets obtained from UCI repository and Caltech-256, while skewness is introduced into the data acquired from Landmine and Nuclear Ban data source. By doing so, we can fully investigate how the proposed algorithm could be affected as the class distribution varies.

In the following studies, we set the number of ex-

¹<http://www.ics.uci.edu/~mllearn/MLRepository.html>

²<http://www.ee.duke.edu/~lcarin/LandmineData.zip>

³<http://www.cs.uu.nl/groups/ADA/icdm08cup/>

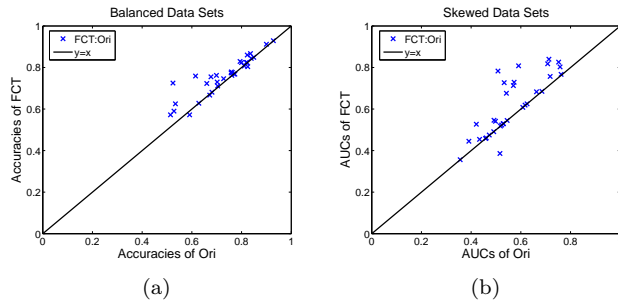


Figure 4: Comparison of Ori and FCT

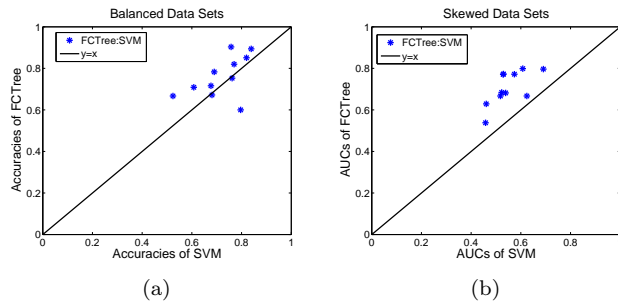


Figure 5: Comparison of FCTree and SVM

panded features in each node equal to the number of original features, i.e. $n_e = n_f$. Six straightforward base and domain independent operators are employed and they are \sqrt{x} , x^2 , $x + y$, $x - y$, $x * y$ and $\exp[-(x - y)^2]$, where x and y represent different features. As one of the baselines for comparison, a two-step (batch process) feature construction method “TFC” is implemented. TFC first enumerates all possible features generated by operators and then selects the top n_f with the highest information gain based on the whole dataset as the obtained feature set. For each dataset, three standard machine learning algorithms, naive Bayes, SVM (with polynomial kernel) and C45 (J48 in Weka) are respectively trained on the original feature set, the feature space obtained through TFC, as well as the feature set constructed by FCTree. Therefore, there are three results for each algorithm. In the tables and figures shown below, “Ori” represents the result achieved on the original feature set. “TFC” is for the result acquired on the feature space obtained through TFC, and “FCT” denotes the result on the feature set constructed by FCTree. It is worth noting that, besides the constructed feature set, the proposed algorithm FCTree itself can also serve as a decision tree classifier, thus we use “FCTree” to represent this learner. All of the classifiers’ performances are measured in accuracy for balanced datasets, and AUC is used instead if the underlying class distribution is skewed. Standard 10-fold cross validation is used to



(a) Not baseball bat



(b) Baseball bat



(c) Baseball bat

Images	(a) vs (b)	(c) vs (b)
Original	0.8879	1.3535
FCTree	1.7278	1.608

(d) Euclidean distances between images

Classifier		Naive Bayes	SVM	J48
Original	(b)	N	Y	Y
	(c)	N	N	N
FCTree	(b)	Y	Y	Y
	(c)	N	Y	Y

(e) Image classification results

Figure 6: Image Examples and Classification Performance

conduct the comparisons, and the algorithm implementations are based on Weka [18].

4.2 Accuracy Comparison on Balanced Datasets

Table 3 summarizes the accuracy of three classifiers achieved on the different feature sets. For each distinct combination of the dataset, learning model and feature set, we highlight the best result in bold. It is evident that, compared to the original feature space (Ori), the feature set constructed by FCTree (FCT) allows different classifiers to achieve much higher accuracy for 23 out of 33 comparisons. A more explicit summary is illustrated in Figure 4(a). Each cross on the plots is the accuracy ratio achieved between the

Table 3: Accuracies on Balanced Datasets: UCI and Caltech-256

Method	NaiveBayes			SVM			J48			FCTree
	Ori	TFC	FCT	Ori	TFC	FCT	Ori	TFC	FCT	
UCI Datasets										
Sonar	0.591	0.576	0.572	0.534	0.567	0.572	0.514	0.567	0.625	<i>0.638</i>
Ionosphere	0.823	0.905	0.860	0.928	0.877	0.929	0.900	0.900	0.912	0.903
Masses	0.822	0.818	0.824	0.800	0.818	0.824	0.825	0.828	0.804	0.743
Transfusion	0.615	0.751	0.759	0.761	0.761	0.762	0.773	0.774	0.768	0.752
Parkinson	0.672	0.614	0.667	0.836	0.761	0.867	0.759	0.774	0.779	<i>0.871</i>
Caltech-256 Datasets										
Caltech1	0.819	0.804	0.824	0.839	0.839	0.844	0.849	0.824	0.849	<i>0.894</i>
Caltech2	0.628	0.639	0.628	0.704	0.690	0.711	0.660	0.686	0.723	<i>0.783</i>
Caltech3	0.524	0.541	0.725	0.681	0.681	0.681	0.528	0.541	0.590	0.672
Caltech4	0.702	0.730	0.730	0.794	0.825	0.829	0.758	0.730	0.774	0.667
Caltech5	0.728	0.750	0.746	0.811	0.820	0.807	0.763	0.768	0.776	<i>0.851</i>
Caltech6	0.677	0.713	0.755	0.769	0.769	0.769	0.699	0.755	0.762	0.716

Table 4: AUCs on Skewed Datasets: Landmine and Nuclear

Method	Naive Bayes			SVM			J48			FCTree
	Ori	TFC	FCT	Ori	TFC	FCT	Ori	TFC	FCT	
Landmine Datasets										
L1	0.752	0.823	0.826	0.491	0.653	0.491	0.356	0.577	0.356	<i>0.847</i>
L2	0.508	0.783	0.783	0.608	0.608	0.608	0.573	0.573	0.729	<i>0.816</i>
L3	0.758	0.803	0.803	0.685	0.685	0.685	0.498	0.541	0.541	<i>0.817</i>
L4	0.708	0.818	0.818	0.625	0.625	0.625	0.591	0.808	0.808	<i>0.878</i>
L5	0.570	0.570	0.713	0.545	0.550	0.545	0.516	0.386	0.386	<i>0.708</i>
Nuclear Datasets										
V	0.762	0.742	0.767	0.530	0.530	0.530	0.542	0.444	0.676	<i>0.773</i>
W	0.713	0.874	0.840	0.529	0.529	0.529	0.533	0.313	0.727	0.771
X	0.663	0.689	0.683	0.518	0.518	0.518	0.492	0.450	0.546	0.667
Y	0.718	0.778	0.757	0.523	0.523	0.523	0.617	0.412	0.619	0.684
Z	0.473	0.539	0.475	0.458	0.458	0.458	0.434	0.434	0.454	0.538
All	0.421	0.561	0.527	0.461	0.461	0.461	0.391	0.391	0.445	<i>0.629</i>

FCT feature set and the original features by a specific algorithm on a dataset. When a cross is above the line “y=x”, it indicates that the accuracy obtained on the FCT feature set is better than that on original features for the corresponding dataset. In particular, when SVM is applied to the UCI data, its accuracy numbers on the FCT feature sets are consistently higher than those obtained using the original features. The overall accuracy is boosted by as much as 9%. When either naive Bayes or C45 serves as the classifier, for 15 out of 22 cases, they perform much better on the feature sets constructed by FCTree. This indicates that the proposed algorithm is not sensitive with respect to different learners. In addition, compared to the feature sets obtained through TFC, different classifiers still tend to achieve higher accuracies on the FCT features.

The win-lose-tie statistics between FCT and TFC is 22-7-4. On the other hand, FCTree outperforms naive Bayes, SVM and C45 on 5 out of 11 datasets. For example, FCTree improves the accuracy at least 5% in Caltech1 dataset compared to other classifiers trained on any feature spaces. This could be ascribed to its good generalization performance and the discriminative power of constructed features as discussed in Section 3.

4.3 An image object classification example Figure 6(a)-(c) show three images selected from the Caltech-256 benchmark dataset [8], where image (a) is from the “AK-47” category, and image (b) and (c) are from the “baseball bat” category. A 177-dimensional color correlogram [9] are extracted as the original image features. Figure 6(d) summarizes that, under the

original feature space, the Euclidean distance between images (b) and (c) is even larger than that between images (a) and (b), even if (a) and (b) belong to the different categories. This is mainly because image (c) has a dominant area of the background clutter, while image (b) does not. This thus leads to dissimilar color-spatial distribution between the two “baseball bat” images. However, after the feature space is converted using FCTree, the most prominent features of these images can be automatically extracted, i.e., the color features on the horizontal center of the images.

Figure 6(e) presents the leave-one-out validation results of three classifiers on the original feature space and the transformed feature space via FCTree. “Y” represents that the image is correctly classified, and “N” indicates the misclassification. The classifiers are trained on all the data from “AK-47” and “baseball-bat” except for the testing images. We observe that in the original feature space, image (c) is misclassified by all three classifiers, and image (b) is misclassified by naive Bayes. However, after the conversion by FCTree, three learners can correctly predict image (b), and only naive Bayes fails on the image (c). This demonstrates that FCTree is able to construct a better feature space for more precise image classification.

4.4 AUC Comparison on Skewed Datasets

Table 4 presents the AUC scores of three classifiers obtained on the different feature sets. It is apparent that, for most of the scenarios, the classifiers can achieve much higher AUCs on the FCT feature sets compared to the original feature space (Ori). Similarly, an explicit summary is illustrated in Figure 4(b). Each cross on the plots is the AUC ratio obtained between the FCT feature set and the original features by a specific algorithm on a dataset. When a cross is above the line “ $y=x$ ”, it indicates that the AUC achieved on the FCT feature set is higher than that on original features for the corresponding dataset. Typically, on the Landmine datasets, when the classifier is naive Bayes, constructed FCT features boost AUC scores at least by 5% with respect to the original feature sets. In addition, classifiers’ performances on the FCT feature sets are comparable to those obtained on the TFC features. The win-lose-tie statistics between FCT and TFC is 10-8-15. This indicates that even though FCT does not perform exhaustive searching, it still be able to capture those discriminative features. Importantly, FCTree outperforms other classifiers on any feature spaces for the Landmine datasets. This suggests that the proposed algorithm can not only construct the useful feature space but also produce a generalized classifier with good performance for skewed distributions.

4.5 Scalability study As discussed in the introduction, exhaustive searching is ineffective due to the infinite feature space. Even if the number of operators is limited, exhaustive searching can also result in combinatorial explosion. In the following, for each dataset, we compare the number of constructed features generated by FCT and TFC. The results are summarized in Figure 7. In general, for most of the datasets, the feature sets generated by FCT are much smaller than those obtained through TFC. This is especially true for those high-dimension datasets. For example, on Caltech-256, the TFC feature sets are at least ten times larger than the feature sets produced by FCT. This provides empirical evidence for the analysis in Section 3.1 that the proposed approach can significantly reduce the computation cost.

4.6 FCTree versus global feature construction

As pointed out in the Introduction, FCTree is a local transform approach. It would be interesting to compare this local transformer to its global counterparts such as SVM with Gaussian kernel. Figure 5 presents the accuracies of FCTree and SVM on the balanced datasets, as well as their AUC scores on the skewed distributions. Each point on the plots is the accuracy or AUC ratio between FCTree and SVM on a specific dataset. When a point is above the line “ $y=x$ ”, it indicates that FCTree performs better than SVM on the corresponding dataset in terms of accuracy or AUC. It is evident that FCTree achieves better accuracies on 8 out of 11 balanced datasets. For all of the skewed distributions, FCTree outperforms SVM without any loss. The overall AUC score is enhanced at least 10% by FCTree.

4.7 Strength of the weighting rule

We conduct one more experiment to study the strength of the weighting rule which allows FCTree to select constructors automatically without the use of manually crafted domain knowledge. For comparison purpose, we implement FCTree without the weighting rule and represent it as FCTW. As compared to FCTree, FCTW randomly selects operators without considering the operators’ weights. C45 (Weka’s J48 implementation) is respectively trained on the two feature spaces generated by FCTree and FCTW. The comparison results on the balanced and skewed distributions are summarized in Figure 8. It is obvious that FCTree outperforms FCTW, and J48’s performance on the FCTree feature set (J48) is better than its performance on the FCTW features (J48w). For the balanced datasets, the win-lose between FCTree and FCTW (or J48 and J48w) is 9-2. On the skewed L2 set, AUC of FCTree is 15% higher

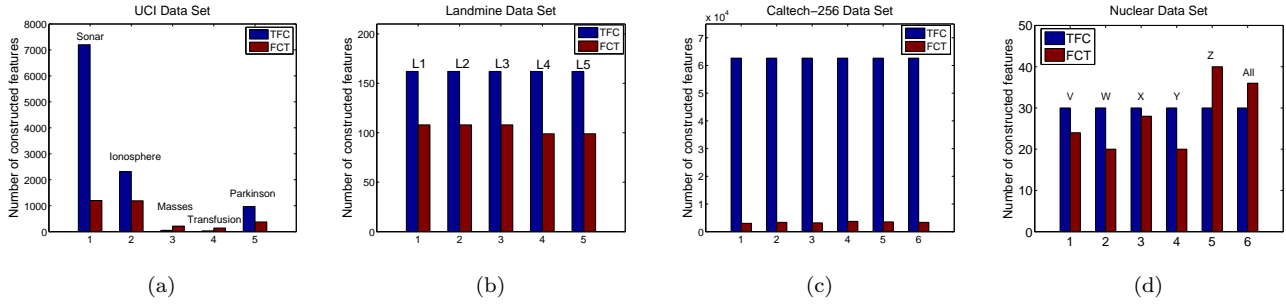


Figure 7: Scalability Analysis

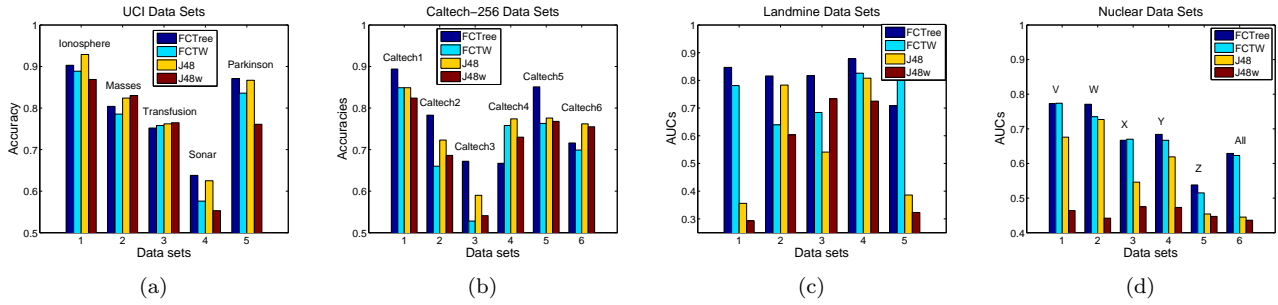


Figure 8: Strength of weighting rule

than that of FCTW. This empirical result suggests that the weighting rule is not trivial but effective.

4.8 Parameter study As shown in Algorithm 2, the number of constructed features in each node n_e should be fixed before applying the proposed algorithm. As the result, we carry out this set of experiments to test the sensitivity and adaptiveness of the proposed algorithms with respect to this parameter. More specifically, we examine the accuracy or AUC learning curves of different classifiers when the number of constructed features n_e changes from n_f to $3.5 \times n_f$, where n_f is the number of original features. The accuracy learning curves on four balanced datasets are presented in Figure 9(a)-(d). It can be observed that, for the same classifier, its accuracy is actually rather stable. The overall accuracy fluctuation of the studied classifiers does not exceed 5%. At the same time, as shown in Figure 9(e)-(h), the AUC scores of different classifiers on the skewed distributions are also not affected much by the varied values of this parameter. The average deviation of AUC is less than 6%. This demonstrates that the proposed algorithm is not sensitive with respect to the different numbers of the constructed features at each node. Most importantly, this property suggests that the proposed method could select a set of compact yet discriminative features.

5 Related Work

State-of-the-art learning techniques are still based on feature vectors. One of the main challenges in classification is to find good feature representation so that an effective classifier can be built from labeled data. Several methods use two-step batch feature construction to extend the original features [10, 14, 17]. For example, the technique proposed in [10] constructs task-relevant discriminant features based on explanation-based interaction of training examples and prior domain knowledge. The work in [14] presents an iterative construction algorithm as a pre-processing step of classification, exhausting all obtained features and then selecting the best ones. However, exhaustive search can be computationally infeasible.

Many other techniques have been proposed for achieving better representations, such as kernel discriminant analysis (KDA) [1] and kernel principal component (KPCA) analysis [16]. More recently, techniques based on frequent pattern mining [4] have been proposed for constructing discriminant features. They are very effective in problems that do not have well-structured feature representations. The proposed approach is related to the one proposed in [4]. While they both employ divide and conquer strategy, the proposed approach introduces feature operators and weighting rules to select operators, that can be adjusted on the fly.

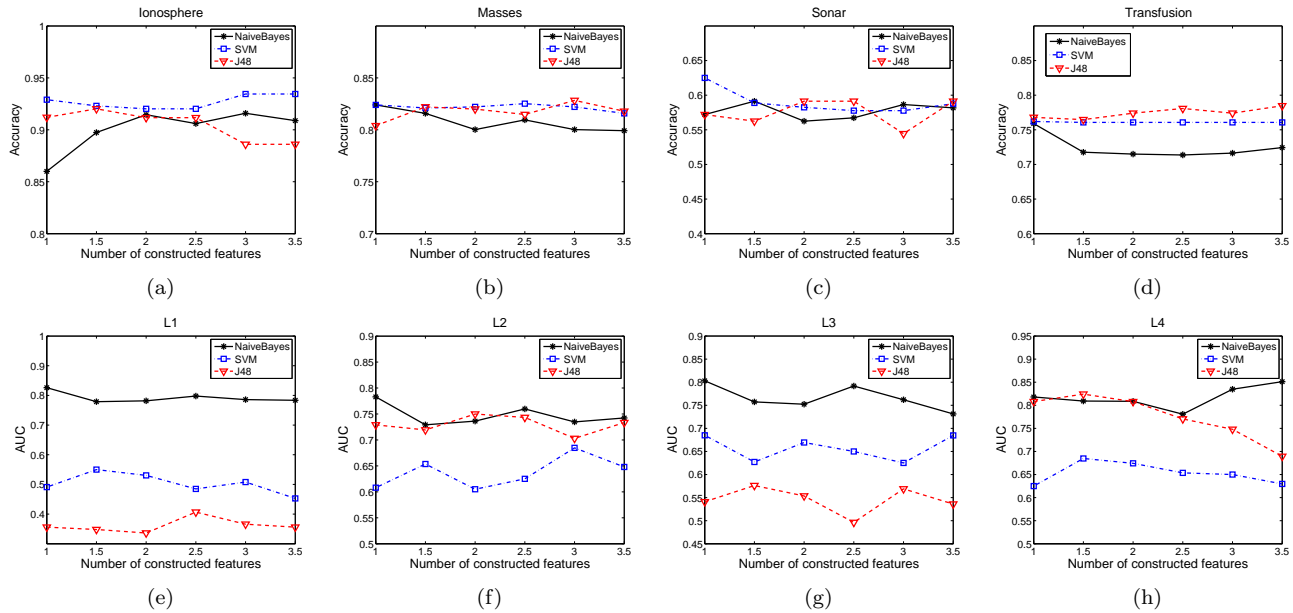


Figure 9: Parameters Analysis

Some previous work avoid exhaustive search using decision tree, such as [13, 15, 12]. The approach in [13] enlarges the attribute set with high level attributes by modifying the initial bias determined on the primitive attributes. But the features constructed by these methods have no generalization guarantee for other classifiers. The framework in [11] improves over [13]. It takes as input a set of classified objects, a set of attributes, and a specification for a set of domain specific constructor functions and produces as output a set of generated features. But it needs the domain information to select constructors.

A number of tree-based feature selection methods has been proposed in the literature, such as [3, 5, 7]. A variable selection technique based on flexible neural trees is proposed in [3], where trees are constructed using genetic programming (GP) and parameters are optimized with the mimetic algorithm (MA). The work presented in [5] is a two-stage method that applies a permutation test for selecting features in a decision tree. The technique described in [7] performs feature selection during decision tree construction using the Separability of Split Value (SSV) criterion. The major difference between FCTree and these tree based methods is that the FCTree algorithm creates new features during feature selection, while these other methods do not. In addition, the Multivariate Decision Tree (MDT) algorithm [2] combines multiple features at each node to split data, but it is simply a classifier, not a feature construction technique.

6 Conclusion

State-of-the-art inductive learners still mainly work on feature vectors. One important challenge is to construct new features using existing feature vector, that has highly discriminative information not obvious in its original feature space. This paper studies how to efficiently and automatically construct highly predictive features that can be generalized over a large number of classifiers. It works by applying a set of local mapping functions on one or a set of features. The significance of new features is evaluated on subspace of examples, not necessarily obvious if evaluated on the complete set of examples. To solve the computational challenge of a possibly infinite search space, we have explored a divide-conquer based decision tree approach. To avoid the use of domain knowledge, the choice of which construction operator to apply is computed dynamically based on their performance via the use of a weighting rule. Formal analysis shows that: (1) the number of selected features is bounded; (2) the weighting rule can effectively assist FCTree to construct highly predictive features without domain related heuristics; (3) it can select best features without running into exhaustive search; (4) the error rate of the classifier built on the new feature space is bounded. Empirical studies have used balanced, unbalanced, normal and high dimensional datasets. The results demonstrate that the proposed method generates new feature space that increases the accuracy and AUC of state-of-the-art classification methods, i.e., Naive Bayes, C45 and SVM polynomial

kernel, by as much as 9% and 28% respectively as compared to the same algorithm trained on the original feature space. FCTree outperforms the batch-based construction method in 22 out of 33 comparisons but reduces the computation cost by at least 10 times.

Acknowledgement

The research of Kun Zhang is supported by an NIH RCMI grant (1G12RR026260-01), and a Louisiana BOR award (LEQSF(2008-11)-RD-A-32) and the research of Jiangtao Ren is supported by the National Natural Science Foundation of China under Grant No. 60703110. Erheng Zhong and Qiang Yang thank the support of the support from Hong Kong CERG grant NSFC/RGC JRS 2009/10.

References

- [1] G. Baudat and F. Anouar. Generalized discriminant analysis using a kernel approach. *Neural Comput.*, 12(10):2385–2404, 2000.
- [2] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. *Mach. Learn.*, 19(1):45–77, 1995.
- [3] Yuehui Chen, Ajith Abraham, and Bo Yang. Feature selection and classification using flexible neural tree. *Neurocomputing*, 70(1-3):305–313, 2006.
- [4] Wei Fan, Kun Zhang, Hong Cheng, Jing Gao, Xifeng Yan, Jiawei Han, Philip Yu, and Olivier Verscheure. Direct mining of discriminative and essential frequent patterns via model-based search tree. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 230–238, New York, NY, USA, 2008. ACM.
- [5] Eibe Frank and Ian H. Witten. Using a permutation test for attribute selection in decision trees. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 152–160, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
- [6] Mostefa Golea, Peter L. Bartlett, Wee Sun Lee, and Llew Mason. Generalization in decision trees and dnf: does size matter? In *NIPS '97: Proceedings of the 1997 conference on Advances in neural information processing systems 10*, pages 259–265, Cambridge, MA, USA, 1998. MIT Press.
- [7] Krzysztof Grabczewski and Norbert Jankowski. Feature selection with decision tree criterion. In *HIS '05: Proceedings of the Fifth International Conference on Hybrid Intelligent Systems*, pages 212–217, Washington, DC, USA, 2005. IEEE Computer Society.
- [8] G. Griffin, A. Holub, and P. Perona. The caltech-256. Technical report, California Institute of Technology, 2007.
- [9] Wynne Hsu, S. T. Chua, and H. H. Pung. An integrated color-spatial approach to content-based image retrieval. In *MULTIMEDIA '95: Proceedings of the third ACM international conference on Multimedia*, pages 305–313, New York, NY, USA, 1995. ACM.
- [10] Shiau Hong Lim, Li-Lun Wang, and Gerald DeJong. Explanation-based feature construction. In *IJCAI'07: Proceedings of the 20th international joint conference on Artificial intelligence*, pages 931–936, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.
- [11] Shaul Markovitch and Dan Rosenstein. Feature generation using general constructor functions. *Mach. Learn.*, 49(1):59–98, 2002.
- [12] Christopher J. Matheus and Larry A. Rendell. Constructive induction on decision trees. In *IJCAI'89: Proceedings of the 11th international joint conference on Artificial intelligence*, pages 645–650, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.
- [13] Giulia Pagallo. Learning dnf by decision trees. In *IJCAI'89: Proceedings of the 11th international joint conference on Artificial intelligence*, pages 639–644, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.
- [14] Selwyn Piramuthu and Riyaz T. Sikora. Iterative feature construction for improving inductive learning algorithms. *Expert Syst. Appl.*, 36(2):3401–3406, 2009.
- [15] Jeffrey C. Schlimmer and Richard H. Granger, Jr. Incremental learning from noisy data. *Mach. Learn.*, 1(3):317–354, 1986.
- [16] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Comput.*, 10(5):1299–1319, 1998.
- [17] Matthew G. Smith and Larry Bull. Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines*, 6(3):265–281, 2005.
- [18] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.