Gradient Boosted Feature Selection

Zhixiang (Eddie) Xu *
Washington University in St. Louis
One Brookings Dr.
St. Louis, USA
xuzx@cse.wustl.edu

Gao Huang
Tsinghua University
30 Shuangqing Rd.
Beijing, China
huang-g09@mails.tsinghua.edu.cn

Kilian Q. Weinberger
Washington University in St. Louis
One Brookings Dr.
St. Louis, USA
kilian@wustl.edu

Alice X. Zheng *
GraphLab
936 N. 34th St. Ste 208
Seattle, USA
alicez@graphlab.com

ABSTRACT

A feature selection algorithm should ideally satisfy four conditions: reliably extract relevant features; be able to identify non-linear feature interactions; scale linearly with the number of features and dimensions; allow the incorporation of known sparsity structure. In this work we propose a novel feature selection algorithm, Gradient Boosted Feature Selection (GBFS), which satisfies all four of these requirements. The algorithm is flexible, scalable, and surprisingly straight-forward to implement as it is based on a modification of Gradient Boosted Trees. We evaluate GBFS on several real world data sets and show that it matches or outperforms other state of the art feature selection algorithms. Yet it scales to larger data set sizes and naturally allows for domain-specific side information.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Miscellaneous; I.5.2 [**Pattern Recognition**]: Design Methodology—*Feature evaluation and selection*

General Terms

Learning

Keywords

Feature selection; Large-scale; Gradient boosting

*Work done while at Microsoft Research

KDD'14, August 24–27, 2014, New York, NY, USA.

ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623635 .

1. INTRODUCTION

Feature selection (FS) [8] is an important problems in machine learning. In many applications, *e.g.*, bio-informatics [21] or neuroscience [12], researchers hope to gain insight by analyzing *how* a classifier can predict a label and what features it uses. Moreover, effective feature selection leads to parsimonious classifiers that require less memory [25] and are faster to train and test [5]. It can also reduce feature extraction costs [29, 30] and lead to better generalization [9].

Linear feature selection algorithms such as LARS [7] are highly effective at discovering linear dependencies between features and labels. However, they fail when features interact in nonlinear ways. Nonlinear feature selection algorithms, such as Random Forest [9] or recently introduced kernel methods [32, 23], can cope with nonlinear interactions. But their computational and memory complexity typically grow super-linearly with the training set size. As data sets grow in size, this is increasingly problematic. Balancing the twin goals of *scalability* and *nonlinear* feature selection is still an open problem.

In this paper, we focus on the scenario where data sets contain a large number of samples. Specifically, we aim to perform efficient feature selection when the number of data points is much larger than the number of features ($n \gg d$). We start with the (NP-Hard) feature selection problem that also motivated LARS [7] and LASSO [26]. But instead of using a linear classifier and approximating the feature selection cost with an l_1-norm, we follow [31] and use gradient boosted regression trees [7] for which greedy approximations exist [2].

The resulting algorithm is surprisingly simple yet very effective. We refer to it as Gradient Boosted Feature Selection (GBFS). Following the gradient boosting framework, trees are built with the greedy CART algorithm [2]. Features are selected sparsely following an important change in the impurity function: splitting on new features is penalized by a cost $\lambda > 0$, whereas re-use of previously selected features incurs no additional penalty.

GBFS has several compelling properties. 1. As it learns an ensemble of regression trees, it can naturally discover nonlinear interactions between features. 2. In contrast to, *e.g.*, FS with Random Forests, it unifies feature selection and classification into a single optimization. 3. In contrast

to existing nonlinear FS algorithms, its time and memory complexity scales as $O(dn)$, where d denotes the number of features dimensionality and n the number of data points[1], and is very fast in practice. 4. GBFS can naturally incorporate pre-specified feature cost structures or side-information, *e.g.*, select bags of features or focus on regions of interest, similar to generalized lasso in linear FS [19].

We evaluate this algorithm on several real-world data sets of varying difficulty and size, and we demonstrate that GBFS tends to match or outperform the accuracy and feature selection trade-off of Random Forest Feature Selection, the current state-of-the-art in nonlinear feature selection.

We showcase the ability of GBFS to naturally incorporate side-information about inter-feature dependencies on a real world biological classification task [1]. Here, features are grouped into nine pre-specified bags with biological meaning. GBFS can easily adapt to this setting and select entire feature bags. The resulting classifier matches the best accuracy of competing methods (trained on many features) with only *a single* bag of features.

2. RELATED WORK

One of the most widely used feature selection algorithms is Lasso [26]. It minimizes the squared loss with l_1 regularization on the coefficient vector, which encourages sparse solutions. Although scalable to very large data sets, Lasso models only linear correlations between features and labels and cannot discover non-linear feature dependencies.

[17] propose the Minimum Redundancy Maximum Relevance (mRMR) algorithm, which selects a subset of the most responsive features that have high mutual information with labels. Their objective function also penalizes selecting redundant features. Though elegant, computing mutual information when the number of instance is large is intractable, and thus the algorithm does not scale. HSIC Lasso [32], on the other hand, introduces non-linearity by combining multiple kernel functions that each uses a single feature. The resulting convex optimization problem aligns this kernel with a "perfect" label kernel. The algorithm requires constructing kernel matrices for all features, thus its time and memory complexity scale quadratically with input data set size. Moreover, both algorithms separate feature selection and classification, and require additional time and computation for training classifiers using the selected features.

Several other works avoid expensive kernel computation while maintaining non-linearity. Grafting [18] combines l_1 and l_0 regularization with a non-linear classifier based on a non-convex variant of the multi-layer perceptron. Feature Selection for Ranking using Boosted Trees [15] selects the top features with the highest relative importance scores. [27] and [9] use Random Forest. Finally, while not a feature selection method, [31] employ Gradient Boosted Trees to learn cascades of classifiers to reduce test-time cost by incorporating feature extraction budgets into the classifier optimization.

3. BACKGROUND

Throughout this paper we type vectors in bold ($\mathbf{x}_i$), scalars in regular math type (k or C), sets in cursive ($\mathcal{S}$) and matrices in capital bold ($\mathbf{F}$) font. Specific entries in vectors or matrices are scalars and follow the corresponding convention.

The data set consists of input vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathcal{R}^d$ with corresponding labels $\{y_1, \ldots, y_n\} \in \mathcal{Y}$ drawn from an unknown distribution. The labels can be binary, categorical (multi-class) or real-valued (regression). For the sake of clarity, we focus on binary classification $\mathcal{Y} \in \{-1, +1\}$, although the algorithm can be extended to multi-class and regression as well.

3.1 Feature selection with the l_1 norm

Lasso [26] combines linear classification and l_1 regularization

$$\min_{\mathbf{w}} \sum_{(\mathbf{x}_i, y_i)} \ell(\mathbf{x}_i, y_i, \mathbf{w}) + \lambda |\mathbf{w}|_1. \quad (1)$$

In its original formulation, $\ell(\cdot)$ is defined to be the squared loss, $\ell(\mathbf{x}_i, y_i, \mathbf{w}) = (\mathbf{w}^\top \mathbf{x}_i - y_i)^2$. However, for the sake of feature selection, other loss functions are possible. In the binary classification setting, where $y_i \in \{-1, +1\}$, we use the better suited log-loss, $\ell(\mathbf{x}_i, y_i, \mathbf{w}) = \log(1+\exp(y_i \mathbf{w}^\top x_i))$ [11].

3.2 The capped l_1 norm

l_1 regularization serves two purposes: It regularizes the classifier against overfitting, and it induces sparsity for feature selection. Unfortunately, these two effects of the l_1-norm are inherently tied and there is no way to regulate the impact of either one.

[33] introduce the *capped* l_1 norm, defined by the elementwise operation

$$q_\epsilon(w_i) = \min(|w_i|, \epsilon). \quad (2)$$

Its advantage over the standard l_1 norm is that once a feature is extracted, its use is not penalized further — *i.e.*, it penalizes using many features does not reward small weights. This is a much better approximation of the l_0 norm, which only penalizes feature use without interfering with the magnitude of the weights. When ϵ is small enough, *i.e.*, $\epsilon \leq \min_i |w_i|$, we can compute the exact number of features extracted with $q_\epsilon(\mathbf{w})/\epsilon$. In other words, penalizing $q_\epsilon(\mathbf{w})$ is a close proxy for penalizing the number of extracted features. However, the capped l_1 norm is not convex and therefore not easy to optimize.

The capped l_1 norm can be combined with a regular l_1 (or l_2) norm, where one can control the trade-off between feature extraction and regularization by adjusting the corresponding regularization parameters, $\mu, \lambda \geq 0$:

$$\min_{\mathbf{w}} \sum_{(\mathbf{x}_i, y_i)} \ell(\mathbf{x}_i, y_i, \mathbf{w}) + \lambda |\mathbf{w}|_1 + \mu q_\epsilon(\mathbf{w}). \quad (3)$$

Here $q_\epsilon(\mathbf{w})$ denotes $[q_\epsilon(\mathbf{w}_1), \ldots, q_\epsilon(\mathbf{w}_d)]$.

4. GRADIENT BOOSTED FEATURE SELECTION

The classifier in Eq. (3) is better suited for feature selection than plain l_1 regularization. However, it is still *linear*, which limits the flexibility of the classifer. Standard approaches for incorporating non-linearity include the kernel learning [22] and boosting [3]. HSIC Lasso [32] uses kernel learning to discover non-linear feature interactions at a price of quadratic memory and time complexity. Our method uses boosting, which is much more scalable.

[1]In fact, if the storage of the input data is not counted, the memory complexity of GBFS scales as $O(n)$.

Boosting assumes that one can pre-process the data with limited-depth regression trees. Let $\mathcal{H}$ be the set of all possible regression trees. Taking into account limited precision and counting trees that obtain identical values on the entire training set as one and the same tree, one can assume $|\mathcal{H}|$ to be finite (albeit possibly large). Assuming that inputs are mapped into $\mathcal{R}^{|\mathcal{H}|}$ through $\phi(\mathbf{x}) = [h_1(\mathbf{x}), \ldots, h_{|\mathcal{H}|}(\mathbf{x})]^\top$, we propose to learn a linear classifier in this transformed space. Eq. (3) becomes

$$\min_{\boldsymbol{\beta}} \sum_{(\phi(\mathbf{x}_i), y_i)} \ell(\phi(\mathbf{x}_i), y_i, \boldsymbol{\beta}) + \lambda|\boldsymbol{\beta}|_1 + \mu q_\epsilon(\boldsymbol{\beta}). \tag{4}$$

Here, $\boldsymbol{\beta}$ is a sparse linear vector that selects trees. Although it is extremely high dimensional, the optimization in Eq. (4) is tractable because $\boldsymbol{\beta}$ is extremely sparse. Assuming, without loss of generalization, that the trees in $\mathcal{H}$ are sorted so that the first T entries of $\boldsymbol{\beta}$ are non-zero, we obtain a final classifier

$$H(\mathbf{x}) = \sum_{t=1}^{T} \beta_t h_t(\mathbf{x}). \tag{5}$$

Feature selection.

Eq. (4) has two penalty terms: plain l_1 norm and capped l_1 norm. The first penalty term reduces overfitting while the second selects features. However, in its current form, the capped l_1 norm selects *trees* rather than features. We therefore have to modify our setup to explicitly penalize the extraction of features.

To model the total number of features extracted by an ensemble of trees, we define a binary matrix $\mathbf{F} \in \{0,1\}^{d \times T}$, where an entry $F_{ft} = 1$ if and only if the tree h_t uses feature f. With this notation, we can express the total weight assigned to trees that extract feature f as

$$\sum_{t=1}^{T} |F_{ft}\beta_t|. \tag{6}$$

We modify $q_\epsilon(\boldsymbol{\beta})$ to instead penalize the actual weight assigned to *features*. The final optimization becomes

$$\min_{\boldsymbol{\beta}} \ell(\boldsymbol{\beta}) + \lambda|\boldsymbol{\beta}|_1 + \mu \sum_{f=1}^{d} q_\epsilon\left(\sum_{t=1}^{T} |F_{ft}\beta_t|\right). \tag{7}$$

As before, if ϵ is sufficiently small ($\epsilon \leq \min_f |\sum_{t=1}^{T} F_{ft}\beta_t|$), we can set $\mu = 1/\epsilon$ and the feature selection penalty coincides exactly with the number of features used.

4.1 Optimization

The optimization problem in Eq. (7) is non-convex and non-differentiable. Nevertheless, we can minimize it effectively (up to a local fixed point) with gradient boosting [7]. Let $\mathcal{L}(\boldsymbol{\beta})$ denote the loss function to be minimized and $\nabla\mathcal{L}(\beta)_t$ the gradient w.r.t β_t. Gradient boosting can be viewed as coordinate descent where we update the dimension with the steepest gradient at every step. We can assume that the set of all regression trees $\mathcal{H}$ is negation closed, *i.e.*, for each $h \in \mathcal{H}$, we also have $-h \in \mathcal{H}$. This allows us to only follow negative gradients and always *increase* $\boldsymbol{\beta}$. Thus there is always a non-negative optimal β. The search for the dimensions t^* with the steepest negative gradient can be formalized as

$$t^* = \operatorname*{argmin}_t \nabla\mathcal{L}(\beta)_t. \tag{8}$$

In the remainder of this section we discuss approximate minimization strategies that does not require iterating over all possible trees.

l_1-regularization.

Since each step of the optimization increases a single dimension of $\boldsymbol{\beta}$ with a fixed step-size $\alpha > 0$, the l_1 norm of $\boldsymbol{\beta}$ can be written in closed form as $|\boldsymbol{\beta}|_1 = \alpha T$ after T iterations. This means that penalizing the l_1 norm of $\boldsymbol{\beta}$ is equivalent to early stopping after T iterations [7]. We therefore drop the $\lambda|\boldsymbol{\beta}|_1$ term and instead introduce T as an equivalent hyper-parameter.

Gradient Decomposition.

To find the steepest descent direction at iteration $T'+1$, we decompose the (sub-)gradient into two parts, one for the loss function $\ell()$, and one for the capped l_1 norm penalty

$$\nabla\mathcal{L}(\boldsymbol{\beta})_t = \frac{\partial \ell}{\partial \beta_t} + \mu \sum_{f=1}^{d} \nabla q_\epsilon\left(\sum_{t=1}^{T'} F_{ft}\beta_t\right). \tag{9}$$

(Hereafter we drop the absolute value around $F_{ft}\beta_t$, since both F_{ft} and β_t are non-negative.) The gradient of $q_\epsilon(\sum_t F_{ft}\beta_t)$ is not well-defined at the cusp when $\sum_t F_{ft}\beta_t = \epsilon$. But we can take the right-hand limit, since β_t never decreases,

$$\nabla q_\epsilon\Big(\sum_{t=1}^{T'} F_{ft}\beta_t\Big) = \begin{cases} F_{ft}, & \text{if } \sum_t F_{ft}\beta_t < \epsilon \\ 0, & \text{if } \sum_t F_{ft}\beta_t \geq \epsilon. \end{cases} \tag{10}$$

If we set $\epsilon = \alpha$, where $\alpha > 0$ is the step size, then $\sum_t F_{ft}\beta_t \geq \epsilon$ if and only if feature f has already been used in a tree from a previous iteration. Let $\phi_f = 1$ indicate that feature f is still *unused*, and $\phi_f = 0$ otherwise. With this notation we can combine the gradients from the two cases and replace $\nabla q_\epsilon\Big(\sum_{t=1}^{T'} F_{ft}\beta_t\Big)$ with $\phi_f F_{ft}$. We obtain

$$\nabla\mathcal{L}(\boldsymbol{\beta})_t = \frac{\partial \ell}{\partial \beta_t} + \mu \sum_{f=1}^{d} \phi_f F_{ft}. \tag{11}$$

Note that $\phi_f F_{ft} = 1$ if and only if feature f is extracted for the first time in tree t. In other words, the second term effectively penalizes trees that use many new (previously not selected) features.

Greedy Tree construction.

With Eq. (11) we can compute the gradient with respect to any tree. But finding the optimal t^* would still require searching all trees. In the remainder of this section, we transform the search for t^* from a search over all possible dimensions t to a search for the best tree h_t to minimize a pre-specified loss function. The new search can be approximated with the CART algorithm [2].

To this end, we apply the chain rule and decompose $\frac{\partial \ell}{\partial \beta_t}$ into the derivative of the loss ℓ w.r.t. the current prediction evaluated at each input $H(\mathbf{x}_i)$ and the partial derivative

Algorithm 1 GBFS in pseudo-code.

```
Input: data {x_i, y_i}, learning rate ε, iterations T.
Initialize predictions H = 0 and selected feature set Ω = ∅.
for t = 1 to T do
    Learn h_t using greedy CART to minimize the impurity function in Eq. (14).
    Update H = H + ε h_t.
    For each feature f used in h_t, set φ_f = 0 and Ω = Ω ∪ f.
end for
Return H and Ω.
```

$\frac{\partial H(\mathbf{x}_i)}{\partial \beta_t}$,

$$\nabla \mathcal{L}(\beta)_t = \sum_{i=1}^{n} \frac{\partial \ell}{\partial H(\mathbf{x}_i)} \frac{\partial H(\mathbf{x}_i)}{\partial \beta_t} + \mu \sum_{f=1}^{d} \phi_f F_{ft}. \quad (12)$$

Note that $H(\mathbf{x}_i) = \boldsymbol{\beta}^\top \mathbf{h}(\mathbf{x}_i)$ is just a linear sum of all $h_t(\mathbf{x}_i)$, the predictions over training data. Thus $\frac{\partial H(\mathbf{x}_i)}{\partial \beta_t} = h_t(\mathbf{x}_i)$. If we let g_i denote the negative gradient $g_i = -\frac{\partial \ell}{\partial H(\mathbf{x}_i)}$, we can reformulate Eq. (12) as

$$h_t = \underset{h_t \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} -g_i h_t(\mathbf{x}_i) + \mu \sum_{f=1}^{d} \phi_f F_{ft}. \quad (13)$$

Similar to [3], we restrict $\mathcal{H}$ to only normalized trees (i.e. $\sum_i h_t^2(\mathbf{x}_i) = 1$). We can then add two constant terms $\frac{1}{2}\sum_i h_t^2(\mathbf{x}_i)$ and $\frac{1}{2}\sum_i g_i^2$ to eq. (13), and complete the binomial equation.

$$h_t = \underset{h_t \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \Big(g_i - h_t(\mathbf{x}_i)\Big)^2 + \mu \sum_{f=1}^{d} \phi_f F_{ft}. \quad (14)$$

This is now a penalized squared loss—an *impurity function*—and a good solutions can be found efficiently via the greedy CART algorithm for learning regression trees [7]. The first term in Eq. (14) encourages feature splits to best match the negative gradient of the loss function, and the second term rewards splitting on features which have already been used in previous iterations. Algorithm 1 summarizes the overall algorithm in pseudo-code.

4.2 Structured Feature Selection

In many feature selection applications, one may have additional qualitative knowledge about acceptable sparsity patterns. Sometimes features can be grouped into bags and the goal is to select as few bags as possible. Prior work on handling structured sparsity include group lasso [10, 20] and generalized lasso [19]. Our framework can easily handle structured sparsity via the feature cost identity function ϕ_f. For example, we can define $\phi_f = 1$ if and only if *no* feature from the same bag as f has been used in the past, and $\phi_f = 0$ otherwise. The moment a feature from a particular bag is used in a tree, all other features in the same bag become "free" and the classifier is encouraged to use features from this bag exclusively until it starts to see diminishing returns.

In the most general setting, we can define $\phi_f : \Omega \to \mathcal{R}_0^+$ as a function that maps from the set of previously extracted features to a cost. For example, one could imagine settings where feature extraction appears in stages. Extracting feature f makes feature g cheaper, but not free. One such application might be that of classifying medical images (e.g., MRI scans) where the features are raw pixels and feature groups are local regions of interest. In this case, $\phi_f(\Omega)$ may reduce the "price" of pixels surrounding those in Ω to encourage feature selection with local focus.

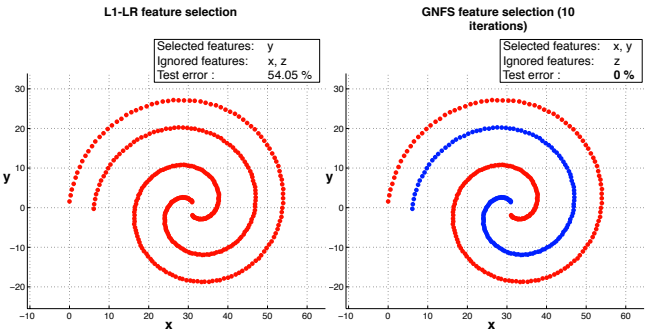

Figure 1: Feature selection and classification performance on a simulated data set. GBFS clearly out-performs the l_1 regularized logistic regression as it successfully captures the nonlinear relations between labels and features.

5. RESULTS

In this section, we evaluate GBFS against other state-of-the-art feature selection methods on synthetic as well as real-world benchmark data sets. We also examine at its capacity for dealing with known sparsity patterns in a bioinformatics application. All experiments were conducted on a desktop with dual 6-core Intel i7 cpus with 2.66GHz, 96 GB RAM, and Linux version 2.6.32.x86_64.

5.1 Synthetic data

Figure 1 illustrates a synthetic binary classification data set with three features. The data is not linearly separable in either two dimensions or three dimensions. However, a good nonlinear classifier can easily separate the data using x and y. The z feature is simply a linear combination of x and y and thus redundant. We randomly select 90% of the instances for training and the rest for testing.

Figure 1 (left panel) illustrates results from *l_1-regularized logistic regression* (L1-LR) [11, 16]. The regularization parameter is tuned on a hold-out set. Although L1-LR successfully detects and ignores the redundant feature z, it also assigns zero weight to x and only selects a single feature y. Consequently, it has poor classification error rate on the test set (54.05%). In contrast, GBFS (Figure 1, right panel) not only identifies the redundant feature z, but also detects that the labels are related to a nonlinear combination of x, y. It selects both x and y and successfully separates the data, achieving 0% classification error.

5.2 Structured feature selection

In many applications there may be prior constraints on the sparsity patterns. Since GBFS can naturally incorporate pre-specified feature structures, we use it to perform

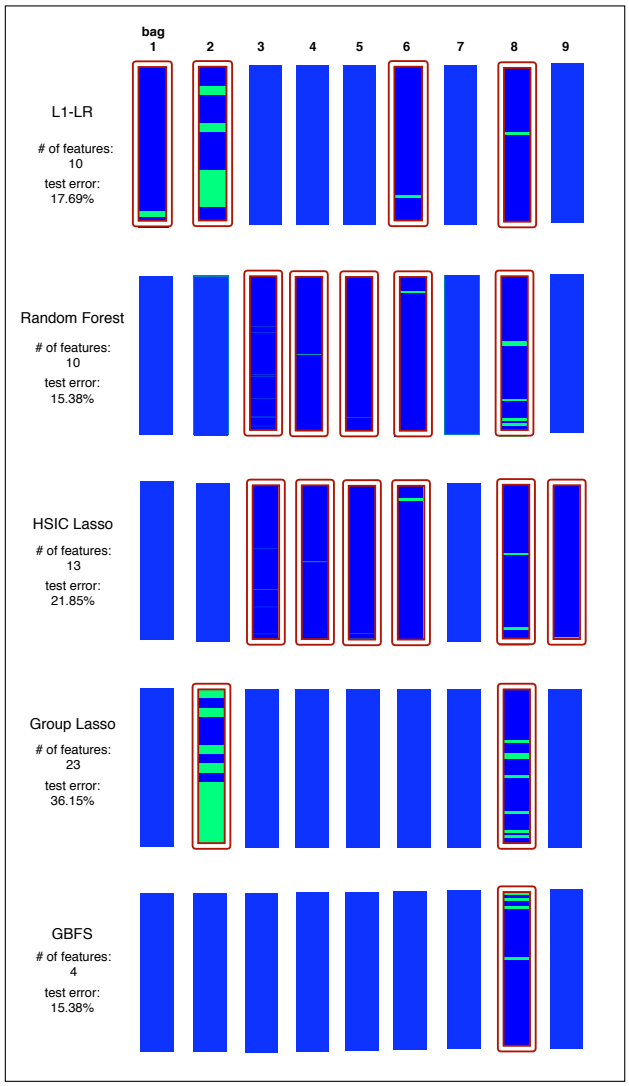

Figure 2: Feature selection on structured feature data set. Selected features are colored in green, and unselected are in blue. The bag is highlighted with a red/white box if at least one of its features is selected by the classifier. (Some bags may require zooming in to make the selected features visible.)

structured feature selection on the Colon data set[2]. In this dataset, 40 tumor and 22 normal colon tissues for 6500 human genes are measured using affymetrix gene chips. [1] select 2000 genes that have the highest minimal intensity across the samples. [13] further analyze these genes and cluster them into 9 clusters/bags by their biological meaning. The task is to classify whether a tissue is normal or tumor. We random split the 62 tissues into 80/20 training and testing datasets, repeated over 10 random splits. We use the feature-bag cost function ϕ_f mentioned in section 4.2 to incorporate this side-information (setting the cost of all features in a bag to *zero* once the first feature is extracted). Feature selection without considering these bag information not only performs and generalizes poorly, but are also difficult to interpret and justify.

Figure 2 shows the selected features from one random split and classification results averaged over 10 splits. Selected features are colored in green, and unselected ones are in blue. A bag is highlighted with a red/white box if at least one of its features is selected by the classifier. We compare against l_1-regularized logistic regression (L1-LR) [11, 16], Random Forest feature selection (RF-FS) [9], HSIC Lasso [32] and Group Lasso [14].

As shown in Figure 2, because GBFS can incorporate the bag structures, it focusses on selecting features in one specific bag. Throughout training, it only selects features from bag 8 (highlighted with a red/white box). This conveniently reveals the association between diseases and gene clusters/bags. Similar to GBFS, Group Lasso with logistic regression can also deal with structured features. However, its l_2 regularization has side effects on feature weights, and thus results in much higher classification error rate 36.15%. In contrast, l_1-regularized logistic regression, Random Forest and HSIC Lasso do not take bag information into consideration. They select scattered features from different bags, making results difficult to interpret. In terms of classification accuracy, GBFS and Random Forest has the lowest test set error rate (15.38%), whereas l_1-regularized logistic regression (L1-LR) and HSIC Lasso achieve error rates of 17.69% and 21.85%, respectively.

There are two reason why GBFS can be accurate with features from only a single bag. First, it is indeed the case that the genes in bag 8 are very predictive for the task of whether the tissue is malignant or benign (a result that may be of high biological value). Second, GBFS does not penalize further feature extraction inside bag 8 while other methods do; since bag 8 features are the most predictive, penalizing against them leads to a worse classifier.

5.3 Benchmark data sets

Data sets.

We evaluate GBFS on real-world benchmark data sets of varying sizes, domains and levels of difficulty. Table 1 lists data set statistics ordered by increasing numbers of training instances. We focus on data sets with a large number of training examples ($n \gg d$). All tasks are binary classification, though GBFS naturally extends to the regression setting, so long as the loss function is differentiable and continuous. Multi-class classification problems can be reduced to binary ones, either by selecting the two classes that are most easily confused or (if those are not known) by grouping labels into two sets.

Baselines.

The first baseline is *l_1-regularized logistic regression (L1-LR)* [11, 16]. We vary the regularization parameter to select different numbers of features and examine the test error rates under each setting.

We also compare against *Random Forest feature selection (RF-FS)* [9], a non-linear classification and feature selection algorithm. The learner builds many full decision trees by bagging training instances over random subsets of features. Features selection is done by ranking features based on their impurity improvement score, aggregated over all trees and all splits. Features with larger impurity improvements are

[2]Available through the Princeton University gene expression project (http://microarray.princeton.edu/oncology/)

data set	pcmac	uspst	spam	isolet	mnist3vs8	adult	kddcup99
#training	1555	1606	3681	6238	11982	32562	4898431
#testing	388	401	920	1559	1984	16282	311029
#features	3289	256	57	617	784	123	122

Table 1: Data sets statistics. Data sets are ordered by the number of training instances.

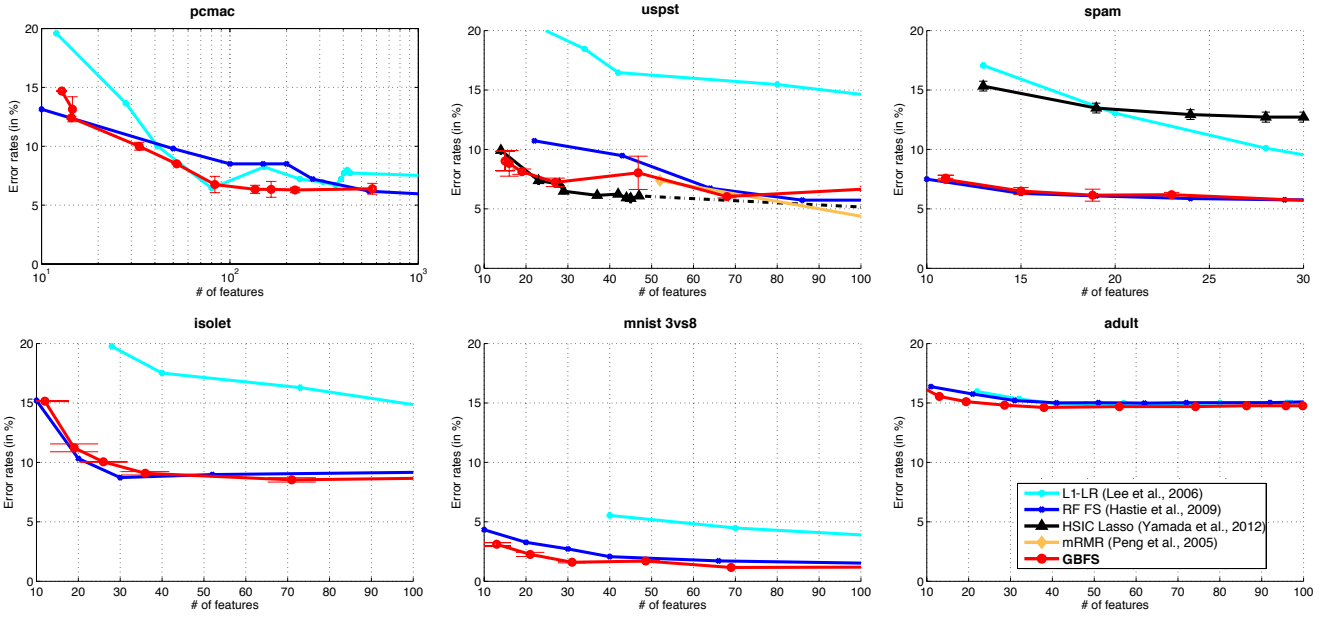

Figure 3: Classification error rates (in %) vs. feature selection performance for different algorithms on small to medium sized data sets.

considered more important. For each data set, we train a Random Forest with 2000 trees and a maximum number of 20 elements per leaf node. After training all 2000 trees, we rank all features. Starting from top of the list, we re-train a random forest with only the top-k features and evaluate its accuracy on the test set. We increase the number of selected features until all features are included.

The next state-of-the-art baseline is *Minimum Redundancy Maximum Relevance (mRMR)* [17], a non-linear feature selection algorithm that ranks features based on their mutual information with the labels. Again, we increase the selected feature set starting from the top of the list. At each stopping point, we train an RBF kernel SVM using only the features selected so far. The hyper-parameters are tuned on on 5 different random 80/20 splits of the training data. The final reported test error rates are based on the SVM trained on the full training set with the best hyper-parameter setting.

Finally, we compare against HSIC Lasso [32], a convex extension to Greedy HSIC [23]. HSIC Lasso builds a kernel matrix for each feature and combines them to best match an ideal kernel generated from the labels. It encourages feature sparsity via an l_1 penalty on the linear combination coefficients. Similar to l_1-regularized logistic regression, we evaluate a wide range of l_1 regularization parameters to sweep out the entire feature selection curve. Since HSIC Lasso is a two steps algorithm, we train a kernel SVM with the selected features to perform classification. Similar to the mRMR experiment, we use cross-validation to select hyper-parameters and average over 5 runs.

To evaluate GBFS, we perform 5 random 80/20 training/validation splits. We use the validation set to choose the depth of the regression trees and the number of iterations (maximum iterations is set to 2000). The learning rate is set to $\epsilon = 0.1$ for all data sets. In order to show the whole error rates curve, we evaluate the algorithm at 10 values for the feature selection trade-off parameter μ in Eq. (7) (i.e., $\mu = \{2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{5}, 2^{7}, 2^{9}\}$).

Error rates.

Figure 3 shows the feature selection and classification performance of different algorithms on small and medium sized data sets. We select up to 100 features except for *spam* ($d = 57$) and *pcmac* ($d = 3289$). In general, l_1-regularized logistic regression (L1-LR), Random Forest (RF-FS) and GBFS easily scale to all data sets. RF-FS and GBFS both clearly out perform L1-LR in accuracy on all data sets due to their ability to capture nonlinear feature-label relationships. HSIC Lasso is very sensitive to the data size (both the number of training instance and the number of features), and only scales to two small data sets (*uspst,spam*). mRMR is even more restrictive (more sensitive to the number of training instance) and thus only works for *uspst*. Both of them run out of memory on *pcmac*, which has the largest number of features. In terms of accuracy, GBFS clearly out-

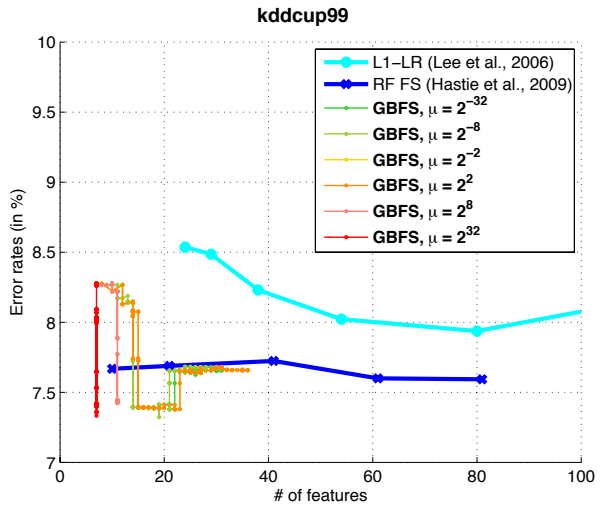

Figure 4: Feature selection and classification error rates (in %) for different algorithms on the large *kddcup99* data set.

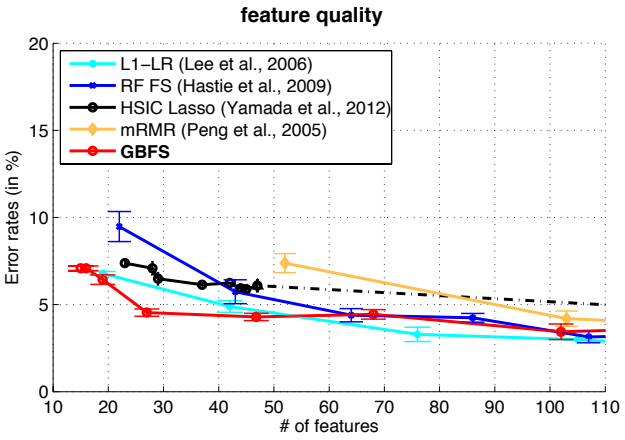

Figure 5: Error rates (in %) of SVM-RBF trained on various feature subset obtained with different features selection algorithms.

performs HSIC Lasso on *spam* but performs slightly worse on *uspst*. On all small and medium datasets, GBFS either outperforms RF-FS or matches its classification performance. However, very different from RF-FS, GBFS is a one step approach that selects features and learns a classifier at the same time, whereas RF-FS requires re-training a classifier after feature selection. This means that GBFS is much faster to train than RF-FS.

Large data set.

The last dataset in Table 1 (*kddcup99*) contains close to 5 million training instances. Training on such large data sets can be very time-consuming. We limit GBFS to $T = 500$ trees with the default hyper-parameters of tree depth $= 4$ and learning rate $= 0.1$. Training Random Forest with default hyper-parameters would take more than a week. Therefore, we limit the number of trees to 100 and the maximum number of instances per leaf node to 500. Feature selection and classification results are shown in Figure 4. For GBFS, instead of plotting the best performing results for each μ, we plot the whole feature selection iteration curve for multiple values of μ. GBFS obtains lower error rates than Random Forest (RF-FS) and l_1 regularized logistic regression (L1-LR) when few features are selected. (Note that due to the extremely large data set size, even improvements of $< 1\%$ are considered significant.)

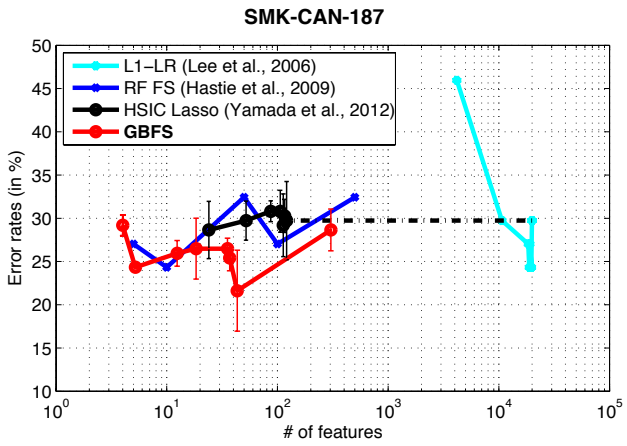

Figure 6: Classification error rates (in %) vs. feature selection performance for different algorithms on a high dimensional ($d \gg n$) data set.

Quality.

To evaluate the quality of the selected features, we separate the contribution of the feature selection from the effect of using different classifiers. We apply all algorithms on the smallest data set (*uspst*) to select a subset of the features and then train a SVM with RBF kernel on the respective feature subset. Figure 5 shows the error rates as a function of the number of selected features. GBFS obtains the lowest error rates in the (most interesting) regions of only few selected features. As more features are selected eventually all FS algorithms converge to similar values. It is worth noting that the linear classifier (L1-LR) slightly outperforms most nonlinear methods when given enough features, which suggests that the uspst digits data requires a nonlinear classifier for prediction but not for feature discovery.

$d \gg n$ scenario.

While GBFS focusses on the scenario where the number of training data points is much larger than the number of features ($n \gg d$), we also evaluate GBFS on a traditional feature selection benchmark data set SMK-CAN-187 [24], which is publicly available from [34]. This binary classification data set contains 187 data points and 19,993 features. We average results over 5 random 80/20 train-test splits. Figure 6 compares the results. GBFS out-performs l_1-regularized logistic regression (L1-LR), HSIC-Lasso and Random Forest feature selection (RF-FS), though by a small margin in some regions.

Computation time and complexity.

Not surprisingly, the linear method (L1-LR) is the fastest by far. Both mRMR and HSIC Lasso take significantly more time than Random Forest and GBFS because they involve either mutual information or kernel matrix computation, which scales as $O(d^2)$ or $O(n^2)$. Random Forest builds full trees, requiring a time complexity of $O(\sqrt{d}n \log n)$ per tree. The dependency on $\sqrt{d}$ is slightly misleading, as the number of trees required for Random Forests is also dependent on the number of features and scales as $O(\sqrt{d})$ itself. In contrast, GBFS only builds limited depth (depth $= 4, 5$) trees, and the computation time complexity is $O(dn)$. The number of iterations T is independent of the number of input features d; it is only a function of how the number of desired features. Empirically, we observe that the two algorithms are comparable in speed but GBFS is significantly faster on data sets with many instances (large n). The training time ranged from several seconds to minutes on the small data sets to about one hour on the large data set *kddcup99* (when Random Forest is trained with only 500 trees and large leaf sizes). Admittedly, the empirical comparison of training time is slightly problematic because our Random Forest implementation is based on highly optimized C++ code, whereas GBFS is implemented in Matlab$^{\text{TM}}$. We expect that GBFS could be made significantly faster if implemented in faster programming languages (*e.g.* C++) with the incorporation of known parallelization techniques for limited depth trees [28].

6. DISCUSSION

This paper introduces GBFS, a novel algorithm for non-linear feature selection. The algorithm quickly train very accurate classifiers while selecting high quality features. In contrast to most prior work, GBFS is based on a variation of gradient boosting of limited depth trees [7]. This approach has several advantages. It scales naturally to large data sets and it combines learning a powerful classifier and performing feature selection into a single step. It can easily incorporate known feature dependencies, a common setting in biomedical applications [1], medical imaging [6] and computer vision [4]. This has the potential to unlock interesting new discoveries in a variety of application domains. From a practitioners perspective, it is now worth investigating if a data set has inter-feature dependencies that could be provided as additional side-information to the algorithm.

One bottleneck of GBFS is that it explores features using the CART algorithm, which has a complexity of $O(dn)$. This may become a problem in cases with millions of features. Although this paper primarily focusses on the $n \gg d$ scenario, as future work we plan to consider improving the scalability with respect to d. One promising approach is to restrict the search to a random subsets of new features, akin to Random Forest. However, in contrast to Random Forest, the iterative nature of GBFS allows us to bias the sampling probability of a feature by its splitting value from previous iterations—thus avoiding unnecessary selection of unimportant features.

We are excited by the promising results of GBFS and believe that the use of gradient boosted trees for feature selection will lead to many interesting follow-up results. This will hopefully spark new algorithmic developments and improved feature discovery across application domains.

7. ACKNOWLEDGMENTS

KQW was supported by NSF grants 1149882 and 1137211. KQW and ZEX were supported by NSF IIS-1149882 and IIS- 1137211. Part of this work was done while ZEX was an intern at Microsoft Research, Redmond.

8. REFERENCES

[1] U. Alon, N. Barkai, D. A. Notterman, K. Gish, S. Ybarra, D. Mack, and A. J. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745–6750, 1999.

[2] L. Breiman. *Classification and regression trees*. Chapman & Hall/CRC, 1984.

[3] O. Chapelle, P. Shivaswamy, S. Vadrevu, K. Weinberger, Y. Zhang, and B. Tseng. Boosted multi-task learning. *Machine learning*, 85(1):149–173, 2011.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[5] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the $l1$-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279. ACM, 2008.

[6] J. A. Etzel, V. Gazzola, and C. Keysers. An introduction to anatomical ROI-based fMRI classification analysis. *Brain Research*, 1282:114–125, 2009.

[7] J. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, pages 1189–1232, 2001.

[8] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[9] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning*. Springer, 2009.

[10] J. Huang, T. Zhang, and D. Metaxas. Learning with structured sparsity. *The Journal of Machine Learning Research*, 12:3371–3412, 2011.

[11] S. Lee, H. Lee, P. Abbeel, and A. Y. Ng. Efficient l1 regularized logistic regression. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 401. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[12] Y. Liu, M. Sharma, C. Gaona, J. Breshears, J. Roland, Z. Freudenburg, E. Leuthardt, and K. Q. Weinberger. Decoding ipsilateral finger movements from ecog signals in humans. In *Advances in Neural Information Processing Systems*, pages 1468–1476, 2010.

[13] S. Ma, X. Song, and J. Huang. Supervised group lasso with applications to microarray data analysis. *BMC bioinformatics*, 8(1):60, 2007.

[14] L. Meier, S. Van De Geer, and P. Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008.

[15] F. Pan, T. Converse, D. Ahn, F. Salvetti, and G. Donato. Feature selection for ranking using boosted trees. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 2025–2028. ACM, 2009.

[16] M. Y. Park and T. Hastie. L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):659–677, 2007.

[17] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1226–1238, 2005.

[18] S. Perkins, K. Lacker, and J. Theiler. Grafting: Fast, incremental feature selection by gradient descent in function space. *The Journal of Machine Learning Research*, 3:1333–1356, 2003.

[19] V. Roth. The generalized lasso. *Neural Networks, IEEE Transactions on*, 15(1):16–28, 2004.

[20] V. Roth and B. Fischer. The group-lasso for generalized linear models: Uniqueness of solutions and efficient algorithms. In *Proceedings of the 25th international conference on Machine learning*, pages 848–855. ACM, 2008.

[21] Y. Saeys, I. Inza, and P. Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[22] B. Schölkopf and A. Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[23] L. Song, A. Smola, A. Gretton, J. Bedo, and K. Borgwardt. Feature selection via dependence maximization. *The Journal of Machine Learning Research*, 98888:1393–1434, 2012.

[24] A. Spira, J. E. Beane, V. Shah, K. Steiling, G. Liu, F. Schembri, S. Gilman, Y.-M. Dumas, P. Calner, P. Sebastiani, et al. Airway epithelial gene expression in the diagnostic evaluation of smokers with suspect lung cancer. *Nature medicine*, 13(3):361–366, 2007.

[25] S. Sra. Fast projections onto $l1$, q-norm balls for grouped feature selection. In *Machine Learning and Knowledge Discovery in Databases*, pages 305–317. Springer, 2011.

[26] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[27] E. Tuv, A. Borisov, G. Runger, and K. Torkkola. Feature selection with ensembles, artificial variables, and redundancy elimination. *The Journal of Machine Learning Research*, 10:1341–1366, 2009.

[28] S. Tyree, K. Weinberger, K. Agrawal, and J. Paykin. Parallel boosted regression trees for web search ranking. In *WWW*, pages 387–396. ACM, 2011.

[29] Z. Xu, M. K., M. Chen, and K. Q. Weinberger. Cost-sensitive tree of classifiers. In S. Dasgupta and D. Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 133–141. JMLR Workshop and Conference Proceedings, 2013.

[30] Z. Xu, M. Kusner, G. Huang, and K. Q. Weinberger. Anytime representation learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1076–1084, 2013.

[31] Z. Xu, K. Weinberger, and O. Chapelle. The greedy miser: Learning under test-time budgets. In *ICML*, pages 1175–1182, 2012.

[32] M. Yamada, W. Jitkrittum, L. Sigal, E. P. Xing, and M. Sugiyama. High-dimensional feature selection by feature-wise non-linear lasso. *arXiv preprint arXiv:1202.0515*, 2012.

[33] T. Zhang. Multi-stage convex relaxation for learning with sparse regularization. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1929–1936. 2008.

[34] Z. Zhao, F. Morstatter, S. Sharma, S. Alelyani, A. Anand, and H. Liu. Advancing feature selection research. *ASU Feature Selection Repository*, 2010.

APPENDIX

A. SUPPLEMENTARY RESULTS

We further extend our experimental results by incorporating more one-vs-one pairs from MNIST data set. We randomly pick 6 one-vs-one pairs from MNIST and evaluate GBFS and other feature selection algorithms. The first baseline is l_1-regularized logistic regression (L1-LR). We vary the regularization parameter to select different number of features and examine the error rates under these different settings. We also compare against *Random Forest feature selection* [9]. Same to the procedure described in section 5, we run Random Forest with 2000 trees and a maximum number of 20 elements per leaf node. After training all 2000 trees, we rank all features. Starting from the most important feature, we re-train a random forest with only selected features and evaluate it on testing set. We gradually include less important features until we include all features. The other two baselines (include mRMR, HSIC-Lasso) do not scale on the MNIST data set.

Figure 7 indicates that GBFS consistently matches Random Forest FS, and clearly out-performs l_1-regularized logistic regression.

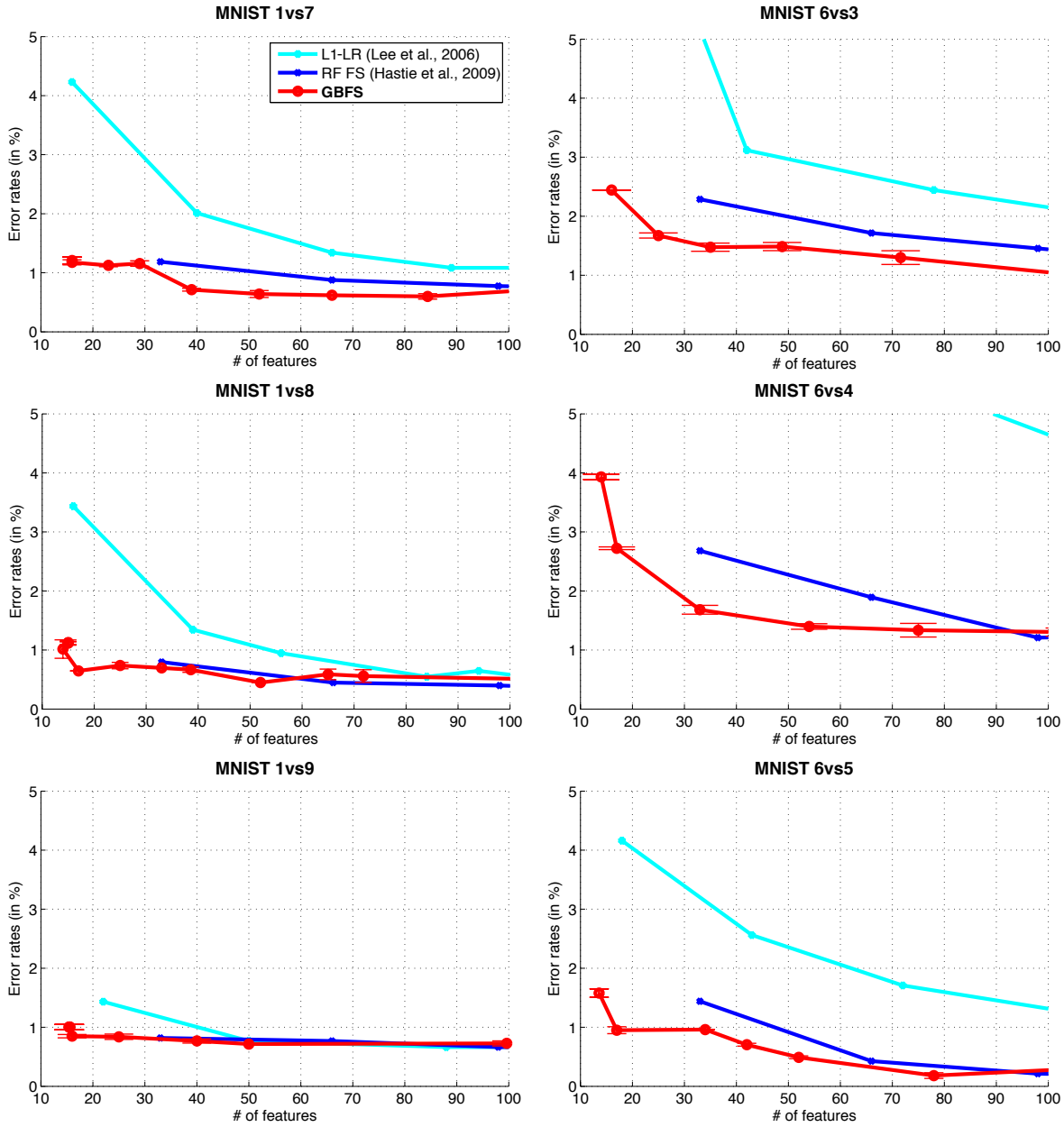

Figure 7: Classification error rates vs. feature selection performance for different algorithms on 6 random chosen pairs of binary classification tasks from MNIST data set.