Fast Flux Discriminant for Large-Scale Sparse Nonlinear Classification

Wenlin Chen
Department of Computer Science and Engineering
Washington University in St. Louis, USA
wenlinchen@wustl.edu

Yixin Chen
Department of Computer Science and Engineering
Washington University in St. Louis, USA
chen@cse.wustl.edu

Kilian Q. Weinberger
Department of Computer Science and Engineering
Washington University in St. Louis, USA
kilian@wustl.edu

ABSTRACT

In this paper, we propose a novel supervised learning method, Fast Flux Discriminant (FFD), for large-scale nonlinear classification. Compared with other existing methods, FFD has unmatched advantages, as it attains the efficiency and interpretability of linear models as well as the accuracy of nonlinear models. It is also sparse and naturally handles mixed data types. It works by decomposing the kernel density estimation in the entire feature space into selected low-dimensional subspaces. Since there are many possible subspaces, we propose a submodular optimization framework for subspace selection. The selected subspace predictions are then transformed to new features on which a linear model can be learned. Besides, since the transformed features naturally expect non-negative weights, we only require smooth optimization even with the ℓ_1 regularization. Unlike other nonlinear models such as kernel methods, the FFD model is interpretable as it gives importance weights on the original features. Its training and testing are also much faster than traditional kernel models. We carry out extensive empirical studies on real-world datasets and show that the proposed model achieves state-of-the-art classification results with sparsity, interpretability, and exceptional scalability. Our model can be learned in minutes on datasets with millions of samples, for which most existing nonlinear methods will be prohibitively expensive in space and time.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications-Data Mining; J.3 [**Computer Applications**]: Life and Medical Sciences

Keywords

classification; interpretability; sparsity; submodularity

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

http://dx.doi.org/10.1145/2623330.2623627.

1. INTRODUCTION

In supervised classification, there are several potentially competing needs. For example in biomedical applications, classifiers often need to be *feature-sparse*, in order to identify leading risk factors for prevention and intervention of critical medical conditions. In addition, the training sets can be very large, thus requiring good *scalability*, but the classifier should also be expressive enough to model *nonlinear* feature interaction. Finally, and possibly most importantly, the classifier must also achieve *high accuracy*. These requirements are by no means specific to biomedical applications. In fact, they are representative for many machine learning application domains. Although several methods excel at various aspects, to date none manages to capture all of them.

Linear classifiers [33] cover many of these needs: they can be highly scalable [8], learn weights that are naturally interpretable and, if paired with ℓ_1 regularization, can also be feature-sparse [7, 29]. Because of these strengths, they are a common choice for machine learning practitioners. However, linear classifiers fall short on at least two of the desired requirements: they cannot learn nonlinear decision boundaries, which inherently limits their accuracy on difficult or lower-dimensional datasets. Further, they cannot discover the relevance of features that are beneficial only through nonlinear interactions with each other.

Nonlinear classifiers can model such feature interactions and are not limited by linear decision boundaries, but they typically suffer from different limitations. For example, deep neural nets [12] are highly nonlinear and scalable, but do not naturally incorporate feature sparsity and the decisions are hard to interpret. Similarly, tree ensembles, such as Random Forests [4], share the same weaknesses. The kernel trick [26] is a popular method to extend linear classifiers to learn nonlinear decision boundaries, but it also removes their natural interpretability and scalability.

To get the best out of both nonlinear methods (nonlinear separability) and linear methods (high efficiency), a recent trend in data mining and machine learning is to replace heavy nonlinear machineries with simpler linear ones, and to achieve high accuracy by introducing certain nonlinearity in **feature mapping**. That is, they map the original input data $\mathbf{x} \in \mathcal{R}^D$ to a feature vector $\phi(\mathbf{x}) \in \mathcal{R}^M$ and then learn a linear hyperplane $\mathbf{w}^T\phi(x)$, $\mathbf{w} \in \mathcal{R}^M$ [17, 22, 24, 30]. A linear classifier, trained on $\phi(\mathbf{x}) \in \mathcal{R}^M$ instead of the "raw" features $\mathbf{x} \in \mathcal{R}^D$, can learn nonlinear decision boundaries in the original input space while maintaining its high scalability. Unfortunately, however, this approach does not also main-

tain its interpretability and feature-sparseness. The classifier weights can no longer be interpreted, which drastically limits the usefulness of this approach for many applications tied to the discovery of feature importance.

As our first contribution, we propose a novel algorithm, which we refer to as Fast Flux Discriminant (FFD). FFD is designed to combine *all* the strengths of linear and nonlinear supervised learning. It maintains the scalability, feature-sparsity and interpretability of linear classifiers, yet can learn non-linear decision boundaries, discover pairwise feature interactions and it matches the high accuracy of state-of-the-art nonlinear models.

At its core, FFD is based on kernel density estimation (KDE) [3]. The ultimate goal of (discriminative) supervised learning is to estimate the conditional label distribution $p(y|\mathbf{x})$, for a label y of an input $\mathbf{x} \in \mathcal{R}^D$. If unlimited labeled data were available, this distribution could be estimated directly with KDE. In practice, however, such a naïve approach does often not work. The curse of dimensionality [3] makes the data requirements of KDE grow *exponentially* with the data dimensionality, thus quickly exceeding any realistic limitations, if the data is sufficiently high dimensional. However, if the dimensionality is low, KDE is very effective.

Chen *et al.* [6] make use of KDE as a feature transformation. They point out that if features do not interact with each other, the conditional label distribution can be decomposed as $\log \frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})} = w_0 + \sum_{d=1}^{D} w_d \log \frac{p(y=1|[\mathbf{x}]_d)}{p(y=-1|[\mathbf{x}]_d)}$, where $[\mathbf{x}]_d$ denotes the d^{th} dimension of $\mathbf{x}$ and the weights w_d are estimated from the data.[1] Similar to Rahimi and Recht [24], this approach learns an explicit feature transformation that enables linear classifiers to learn non-linear decision boundaries. Unfortunately, it cannot discover non-linear feature interactions.

In this paper we follow this insight but *relax* the restriction that features cannot interact with each others. Instead, we assume that features *can* interact, but their interactions are limited to $r \ll D$ features. Let A denote a set or "bag" of features, with $|A| \leq r$, and let $[\mathbf{x}]_A$ denote the *shortened* input vector $\mathbf{x}$ with only those features in A. We assume that r is small enough, so that $p(y|[\mathbf{x}]_A)$ can still be computed fast and accurately via KDE. FFD learns a new representation $\mathbf{x} \rightarrow \Phi(\mathbf{x})$, where each dimension of$\Phi(\mathbf{x})$ corresponds to the conditional label distribution $p(y|[\mathbf{x}]_A)$ for some feature bag A. A *linear* classifier learned on the data$\Phi(\mathbf{x})$ can learn nonlinear decision boundaries and its weights indicate which feature "bags" are most important. If in addition the classifier is regularized to be sparse, we can identify which feature bags are in fact *sufficient* to make accurate predictions. Because KDE is highly non-linear and approximates the true conditional distribution, FFD reliably learns the within-bag feature interactions.

One immediate challenge with this setup is the exponentially growing number, $\binom{D}{r}$, of possible sets A. How can we identify which such sets of features to include in our representation? Our second contribution is to formulate this as a tractable optimization problem. We propose a novel optimization framework which maximizes the similarity coverage and minimizes redundancy of the selected feature set. Moreover, the optimization formulation contains cardinality penalty for sparsity and interpretability. We formulate this selection problem as a combinatorial optimization with a submodular objective. Previous work [9] proves that for this category of problems, there exists a 1/3-approximation bound in the worst case.

To further promote feature sparsity of the FFD model, we also employ ℓ_1 regularization. A disadvantage of typical ℓ_1 regularization is that it is not differentiable everywhere and requires non-smooth optimization techniques such as sub-gradient descent. However, since the KDE features in FFD model conditional label distribution, their weights are naturally expected to be non-negative. Based on this key insight, we add non-negativity constraints to the training objective and make it a *smooth* optimization problem even with ℓ_1 regularization. This enables us to employ fast smooth optimization algorithms.

Finally, we carry out extensive experiments on real-world datasets and show that the proposed FFD model achieves state-of-the-art classification results with exceptional scalability. FFD can be learned in minutes on datasets with millions of samples, for which other nonlinear methods will be prohibitively expensive in space and time. FFD also demonstrates interpretability and results in very sparse models.

This paper is organized as follows. In Section 2, we present the notations and preliminaries. We describe the proposed FFD model in Section 3. We discuss related work in Section 4 and present experimental results in Section 5. Section 6 gives conclusions.

2. PRELIMINARIES

In this section, we introduce the notations and review the density-based logistic regression (DLR) model [6], which is highly related to the proposed model.

Assume we are given a dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}$, $i = 1, \cdots, N$, $\mathbf{x}_i \in \mathcal{R}^D$ where D is the number of attributes[2], and the label $y_i \in \{-1, 1\}$. Let the input vector be $\mathbf{x}_i = ([\mathbf{x}_i]_1, \cdots, [\mathbf{x}_i]_D)$.

$\mathcal{D}$ can be further partitioned into two datasets $\mathcal{D}_1$ and $\mathcal{D}_{-1}$, which include all the data points whose labels are $y = 1$ and $y = -1$, respectively.

In the training phase, the DLR model first maps each training sample to a new feature space in a dimension-wise manner, i.e. $\mathbf{x} \rightarrow \mathbf{\Phi}(\mathbf{x})$ where

$$\mathbf{\Phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \cdots, \phi_D(\mathbf{x})).$$

Similar to logistic regression, DLR models the conditional probability of y given a sample $\mathbf{x}$ by a sigmoid function on the transformed features.

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + \exp\left(-\left(\mathbf{w}\mathbf{\Phi}(\mathbf{x})\right)\right)} \tag{1}$$

where the parameter $\mathbf{w}$ can be learned via maximum likelihood estimation or equivalently the empirical risk minimization with logistic loss:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^{N} \left(1 + e^{-y_i(\mathbf{w}\Phi(\mathbf{x}_i))}\right) + \lambda \|\mathbf{w}\|_2 \tag{2}$$

[1]This approximation is exact for $w_d = 1$ if the features are label conditionally independent, which is also often referred to as the Naïve Bayes assumption [3].

[2]For simplicity, here we only discuss datasets with numerical features. The proposed method also applies to datasets with categorical and mixed features.

If we use the original attributes without feature mapping, i.e. $\phi_d(\mathbf{x}) = [\mathbf{x}]_d$ for $d = 1, \cdots, D$, Eq. (1) is the original LR.

Unlike LR, DLR introduces a nonlinear feature mapping based on the following rationale. First, Eq. (1) can be rewritten as:

$$\mathbf{w}\mathbf{\Phi}(\mathbf{x}) = \ln \frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})} \tag{3}$$

Assuming conditional independence of the attributes given the class label y, it can be shown that [6]

$$\ln \frac{p(y=1|\mathbf{x})}{p(y=-1|\mathbf{x})} = w_0 + \sum_{d=1}^{D} \left(\ln \frac{p(y=1|[\mathbf{x}]_d)}{p(y=-1|[\mathbf{x}]_d)} \right) \tag{4}$$

where $w_0 = (D-1) \ln \frac{p(y=1)}{p(y=-1)}$. Comparing (3) with (4) and setting $\mathbf{w} = 1$, we can naturally derive the following feature transformation used by DLR. For each dimension d, the DLR feature mapping is:

$$\phi_d(\mathbf{x}) = \ln \frac{p(y=1|[\mathbf{x}]_d)}{p(y=-1|[\mathbf{x}]_d)}, \tag{5}$$

which is a logit transformation on the conditional probability of y given a single feature $[\mathbf{x}]_d$.

In order to compute the features $\phi_d(\mathbf{x})$ in (5), DLR estimates $p(y|[\mathbf{x}]_d)$ by treating categorial and numerical attributes in different ways.

- If $[\mathbf{x}]_d$ is a categorical attribute, $p(y=1|[\mathbf{x}]_d)$ is estimated by counting the proportion of samples with label $y = 1$ among all the samples whose d^{th} attribute is $[\mathbf{x}]_d$. Let $\mathcal{D}_{[\mathbf{x}]_d}$ be the set of of samples in $\mathcal{D}$ whose d^{th} attribute equals to $[\mathbf{x}]_d$. The DLR estimate is

$$\phi_d(\mathbf{x}) = \ln \frac{|\mathcal{D}_1 \bigcap \mathcal{D}_{[\mathbf{x}]_d}|}{|\mathcal{D}_{-1} \bigcap \mathcal{D}_{[\mathbf{x}]_d}|}. \tag{6}$$

- If $[\mathbf{x}]_d$ is a numerical attribute, then DLR calculates $\phi_d(\mathbf{x})$ by kernel density estimation (KDE) [3]. The DLR estimate is

$$\phi_d(\mathbf{x}) = \ln \frac{\sum_{i \in \mathcal{D}_1} \exp(-\frac{([\mathbf{x}]_d - [\mathbf{x}_i]_d)^2}{h_d^2})}{\sum_{i \in \mathcal{D}_{-1}} \exp(-\frac{([\mathbf{x}]_d - [\mathbf{x}_i]_d)^2}{h_d^2})}, \tag{7}$$

  where h_d is a parameter called the kernel bandwidth.

In the training phase, DLR first computes the feature mapping and then calls a standard LR package to learn $\mathbf{w}$. In the testing phase, given a testing sample $\mathbf{x}$, the DLR model first transforms it to $\mathbf{\Phi}(\mathbf{x})$ via KDE or counting depending on the feature type. The conditional probability of y given $\mathbf{x}$ is then calculated by Eq. (1).

Though DLR offers good interpretability as it assigns a weight to each original dimension, it has some serious drawbacks. 1) DLR has high training and testing complexity. Although DLR is more efficient in its training time, its feature computation is still expensive. For a dataset with N samples and D dimensions, DLR requires $\mathrm{O}(DN^2)$ time for training and $\mathrm{O}(DN)$ time for testing one single sample. Such a testing cost is the same as kernel SVM, making it too expensive for applications where extensive testings are required. 2) DLR generates dense vectors and does not offer sparsity. 3) DLR assumes conditional independence of each dimension give the class label, which is often violated in practice and may suffer from high correlation between dimensions.

3. FAST FLUX DISCRIMINANT (FFD)

In this section, we propose our FFD model. Like DLR, FFD also performs a feature mapping based on density estimation and then learns a linear machine after the feature mapping. However, FFD is significantly different from DLR and offers a few salient advantages. It does not assume conditional independence and is able to capture the correlation between features. Also, it preserves the interpretability and explicitly promotes sparsity of the model. Moreover, the learning process of FFD is far more efficient than DLR, leveraging on the fast computing of low-dimensional density estimation via histogram estimation.

In summary, the main steps of FFD include the following.

1. Subspace feature mapping, which generates features based on non-parametric kernel density estimation.
2. Submodular subspace selection, which selects subspace features based on submodular optimization.
3. Model training, which learns a sparse and interpretable linear machine with smooth ℓ_1 regularization.

All the above steps are designed to be highly efficient and capable of scaling to large data. Below, we discuss the main components of FFD before putting them together.

3.1 Subspace feature mapping

This step generates features that can effectively model the conditional probability $p(y|\mathbf{x})$, which enables the nonlinear separability of the model. It is based on non-parametric density estimation which does not make any parametric assumption of the data distribution and is particular suitable for large data since it can make full use of all the data samples. Here, we first describe the histogram-based density estimation in general for the full feature space, which is unrealistic due to the curse of dimensionality. But the same idea can be applied to and is very efficient for low-dimensional subspaces. We then combine the predictions from all subspaces via a linear model and generate nonlinear classification decision boundary.

First, each dimension is divided into equal-length bins. Let b_d be the number of bins for the d^{th} dimension. If $[\mathbf{x}]_d$ is categorical, b_d is always the number of categories for this feature. If $[\mathbf{x}]_d$ is numerical, b_d is a parameter we need to set.

In the training phase, given the training dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}$, $i = 1, \cdots, N$, we assign each training sample to the corresponding bin. If $[\mathbf{x}_i]_d$ is a categorical feature, $B([\mathbf{x}_i]_d)$, the bin index for dimension d is the category index of $[\mathbf{x}_i]_d$. For a numerical feature, suppose the bins of the d^{th} dimension start at

$$start_d = \min_{k=1..N} \{[\mathbf{x}_k]_d\}$$

and end at

$$end_d = \max_{k=1..N} \{[\mathbf{x}_k]_d\}.$$

The bin length l_d is given by:

$$l_d = \frac{end_d - start_d}{b_d} \tag{8}$$

Let $B([\mathbf{x}_i]_d)$ be the bin index for $[\mathbf{x}_i]_d$. We have that

$$B([\mathbf{x}_i]_d) = \frac{[\mathbf{x}_i]_d - start_d}{l_d} \tag{9}$$

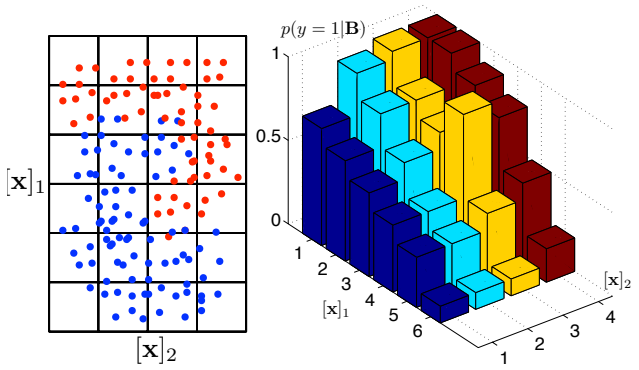

Figure 1: Histogram estimation for a 2-D subspace. Left: a snapshot of a 2-D grid. Right: The proportion of $y = 1$ in each grid cell based on counting.

Each data sample $\mathbf{x}_i$ corresponds to a grid cell[3] (a vector of bin indices):

$$B(\mathbf{x}_i) = (B([\mathbf{x}_i]_1), \cdots, B([\mathbf{x}_i]_D)).$$

After assigning each training sample to the corresponding grid cell, a naive histogram-based density method would estimate the the probability of $p(y|\mathbf{x})$ by counting the proportion of samples with different labels in grid cell $B(\mathbf{x})$. This naive histogram-based density estimation is efficient and does not make any assumption on the distribution of the underlying data. However, if the number of training samples is not enough, this estimation would suffer from huge bias and variance. In addition, the number of grid cells grows exponentially with the number of dimensions, leading to the curse of dimensionality. As a result, this simple histogram method is not practical.

To overcome this problem, instead of directly modeling $p(y|\mathbf{x})$ by density estimation for the whole space, FFD expresses $p(y|\mathbf{x})$ by a number of density estimation for low dimensional subspaces. Each subspace contains a small number (less than r) of features. In essence, we assume that features can interact with each other, but their interactions are limited to $r \ll D$ features. We assume that r is small enough, so that the density estimation for r-dimensional subspaces can still be computed fast. For example, Figure 1 shows the histogram-based density estimation for a 2-dimensional subspace. This subspace is discretized into a grid as shown on the left subfigure. The proportion of training samples with $y = 1$, i.e. $p(y = 1|\mathbf{B})$, is then computed for each grid cell by simple counting as shown on the right subfigure.

To combine the density estimations from all subspaces, we convert the result from each subspace to a new feature and then apply a linear model on these new features. Specifically, FFD learns a new representation $\mathbf{x} \to \Phi(\mathbf{x})$, where

$$\Phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \cdots, \phi_M(\mathbf{x}))$$

is a vector of M subspace features. For each $m = 1, \cdots, M$, FFD uses the following feature

$$\phi_m(\mathbf{x}) = \ln \frac{p(y = 1|[\mathbf{x}]_{A_m})}{p(y = -1|[\mathbf{x}]_{A_m})}, \tag{10}$$

[3]In this paper, the meanings of "bin" and "grid cell" are interchangeable. We often refer to "bin" in the context of a single dimension and "grid cell" otherwise

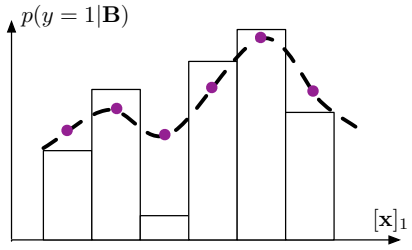

Figure 2: Kernel smoothing for a 1-D histogram. The height of each bin is the proportion of $y = 1$ in each bin based on counting. The purple dots are the new $p(y = 1|\mathbf{B})$ for each bin after kernel smoothing.

where $A_m \subset \{1, \cdots, D\}$, $|A_m| \leq r$, is a subset of feature dimensions and $[\mathbf{x}]_{A_m} = ([\mathbf{x}]_{d \in A_m})$ are $\mathbf{x}$'s values in the dimensions included in A_m.

Following the design of logistic regression, FFD models the following probability:

$$p(y = 1|\mathbf{x}) = \frac{1}{1 + \exp\left(-\sum_{m=1}^{M} w_m \phi_m(\mathbf{x})\right)}. \tag{11}$$

3.2 Kernel smoothing for histogram

FFD estimates $p(y = 1|[\mathbf{x}]_{A_m})$ via histogram-based density estimation. Since the cardinality $|A_m| \leq r$ where $r \ll D$, it avoids the curse of dimensionality.

For relatively small datasets, the histogram estimation for the subspace specified by A_m is still unstable. First, if the length of bins is too small, there would be few data samples in each bin, resulting in inaccurate estimation. Second, the histogram estimation could be non-smooth for neighboring bins.

To address these issues, we propose to use a *bin kernel smoothing* technique, which allows the bins to affect each other according to their mutual kernel. Suppose $\mathbf{B} = (B_1, \cdots, B_{|A_m|})$ is a grid cell in the subspace specified by the dimensions in A_m and denote G_{A_m} as the set of all the grid cells in this subspace. We have the following smoothed estimate for grid cell $\mathbf{B}$

$$p(y|\mathbf{B}) = \frac{\sum_{\mathbf{B}' \in G_{A_m}} n_{\mathbf{B}'}(y) K^{bin}(\mathbf{B}, \mathbf{B}')}{\sum_{\mathbf{B}' \in G_{A_m}} N_{\mathbf{B}'} K^{bin}(\mathbf{B}, \mathbf{B}')} \tag{12}$$

where $n_{\mathbf{B}'}(y)$ is the the number of training samples with label y in $\mathbf{B}'$, and $N_{\mathbf{B}'}$ is the total number of training samples in $\mathbf{B}'$. The kernel between two grid cells $\mathbf{B}$ and $\mathbf{B}'$ is a Guassian kernel given by

$$K^{bin}(\mathbf{B}, \mathbf{B}') = \exp\left(-\sum_{d \in A_m} \frac{(B_d - B'_d)^2 l_d^2}{2h_d^2}\right) \tag{13}$$

where $h_d > 0$ is a parameter called the *bandwidth* of the kernel density function. A popular rule of thumb [27] for deciding the value of h_d is as follow:

$$h_d^* = 1.06 \sigma_d N^{-1/5}, \tag{14}$$

where σ_d is the standard deviation of the training samples on d^{th} dimension. Figure 2 illustrates the kernel smoothing for a 1-dimensional histogram. As we can observe, $p(y = 1|\mathbf{B})$ in the third bin is originally very low due to the lack of data.

However, after kernel smoothing, this value gets higher and the overall histogram becomes smoother.

In essence, the proposed bin kernel smoothing is an approximation of KDE by discretizing each dimension into bins and treating all samples in the bin as located at the center of the bin. Not surprisingly, we can show that the approximation error approaches zero as the discretization is fine enough.

PROPOSITION 1. *As $b_d \to \infty$, the conditional probability estimated by (12) approaches the result by the Nadaraya-Watson estimator [3]*

$$\hat{p}(y=1|[\boldsymbol{x}]_{A_m}) = \frac{\sum_{i\in\mathcal{D}_1} K_{A_m}(\boldsymbol{x},\boldsymbol{x}_i)}{\sum_{i=1}^{N} K_{A_m}(\boldsymbol{x},\boldsymbol{x}_i)} \tag{15}$$

where $\mathcal{D}_1$ is the set of training samples with label $y=1$ and $K_{A_m}(\boldsymbol{x}_1,\boldsymbol{x}_2)$ is a Guassian kernel function on variables specified by A_m, namely,

$$K_{A_m}(\boldsymbol{x}_1,\boldsymbol{x}_2) = \exp\left(-\sum_{d\in A_m}\frac{([\boldsymbol{x}_1]_d-[\boldsymbol{x}_2]_d)^2}{2h_d^2}\right). \tag{16}$$

The size of memory required to store all histogram information is $O\left(\sum_{m=1}^{M} b_{A_m}\right)$ where $b_{A_m} = \prod_{d\in A_m} b_d$ is the total number of grid cells required for the subspace specified by A_m. Since $|A_m|$ is always very small (e.g. less than 3), the memory cost will not be large. One advantage of FFD is that the size of the model does not increase as the training dataset gets larger. This is different from other popular non-parametric models such as RBF-SVM and KNN. For example, in RBF-SVM all the support vectors should be stored. As the training set gets larger, the number of support vectors will also increase.

In addition, in the testing phase, the kernel matrix between support vectors and testing samples needs to be computed, resulting in $O(N_tN_s)$ running time where N_t is the size of the testing data and N_s is the number of support vectors. In contrast, FFD has a linear test time $O(N_t)$, since for a given testing sample FFD only needs to retrieve the value stored in the histogram in order to compute ϕ_m.

3.3 Submodular subspace selection

Let U be the ground set containing all subspace candidates, i.e. all $A_m \subset \{1,\cdots,D\}$ such that $|A_m| \le r$. The cardinality of U is $\sum_{k=1}^{r}\binom{D}{k}$. With so many candidates, one key challenge is to determine which subspaces in U to choose from. Here we propose a combinatorial optimization framework to address this problem. Suppose $S \subseteq U$ is the set of subspaces we select, we propose to find S such that

$$\underset{S}{\text{maximize}} \sum_{i\in S}\sum_{j\in U-S} c_{i,j} - \alpha \sum_{i,j\in S, i\neq j} c_{i,j} + \beta\sum_{i\in S} a_i - \mu|S|^2 \tag{17}$$

where $c_{i,j}$ is the Pearson correlation between ϕ_i and ϕ_j based on A_i and A_j, respectively. a_i is the training accuracy of the subspace estimation for G_{A_i}, which can be computed efficiently by simply checking the number of mislabeled samples in all grid cells of G_{A_i}. (α,β,μ) are all non-negative hyper-parameters.

The four components in (17) correspond to four different goals, respectively. 1) The first term, which is a cut function [10], maximizes the similarity coverage of the set U so that set S is a good representative of U. 2) The second term minimizes the pair-wise correlations within S to reduce the redundancy and relieve the problem of highly correlated features and co-linearity. 3) The third term maximizes the overall accuracy of histogram estimation of the selected subspaces. 4) The fourth term minimizes the cardinality of S for sparsity.

The maximization of (17) is a NP-hard problem. However, we show that (17) is a submodular maximization problem and good approximation bound can be achieved. We first introduce the concept of submodular set functions.

Definition 1. [10] Suppose U is the ground set, a set function $f: 2^U \to \mathcal{R}$ is submodular if it satisfies the property of diminishing return: for every $A, B \in U$, $A \subseteq B$, and every $e \in U - B$, we have that

$$f(A\cup e) - f(A) \ge f(B\cup e) - f(B). \tag{18}$$

If equality always holds in (18), f is modular.

It has been shown that maximizing a submodular function without any constraints can achieve a 1/3-approximation bound using a deterministic local search (DLS) algorithm [9, 13]. That is,

$$f_{obj}(S_g) \ge \frac{1}{3} f_{obj}(S^*),$$

where f_{obj} is the objective function in (17), S_g is the solution by the DLS algorithm, and S^* is the optimal solution. Note that this lower bound only occurs in the worst case. In practice, the performance is typically much better.

Now we prove the submodularity of (17).

THEOREM 1. *The objective function in (17) is submodular if $c_{i,j} \ge 0, \forall i,j \in U$.*

PROOF. Let

$$f_c(S) = \sum_{i\in S}\sum_{j\in U-S} c_{i,j} - \alpha \sum_{i,j\in S, i\neq j} c_{i,j}. \tag{19}$$

It has been shown that $f_c(S)$ is submodular if $c_{i,j} \ge 0, \forall i,j \in U$ [20]. Moreover, it is easy to verify that $\beta\sum_{i\in S} a_i$ is a modular function and that $-\mu|S|^2$ is submodular function according to Definition 1. Since submodularity is preserved under non-negative linear combination of submodular functions [10], we see that the the objective function in (17) is submodular. □

In practice, we find that most $c_{i,j}$ are non-negative, since each ϕ_m is an indicator of the label y and they are not likely to have negative correlation. For completeness, we have the following lemma to help guarantee the submodularity of (17).

LEMMA 1. *Let $\gamma = -\min\{\min_{i,j}\{c_{i,j}\}, 0\}$, then $f_c(S) - \gamma(1+\alpha)|S|^2$ is submodular where $f_c(S)$ is defined in (19).*

PROOF. Let e_S be the $|U| \times 1$ indicator vector of set S where U is the ground set, and $e_S(i) = 1$ if $A_i \in S$ and 0 otherwise [4] (so e_U is a vector whose elements are all 1). Let C be the correlation matrix of all elements in U where its element (i,j) is $c_{i,j}$. We rewrite the function in matrix form as follows

$$\begin{aligned} f_c(S) &= e_S^\top C(e_U - e_S) - \alpha e_S^\top (C-I)e_S \\ &= e_S^\top C e_U + \alpha e_S^\top I e_S - (\alpha+1)e_S^\top C e_S \\ &= e_S^\top C e_U + \alpha|S| - (\alpha+1)e_S^\top C e_S \end{aligned} \tag{20}$$

[4] $e_S(i)$ is the i^{th} element in e_S

Suppose E is a $|U| \times |U|$ matrix with all elements being 1. It is easy to show that $|S|^2 = e_S^\top E e_S$. Thus, we have that

$$\begin{aligned} & f_c(S) - \gamma(1+\alpha)|S|^2 \\ =& e_S^\top C e_U + \alpha|S| - (\alpha+1)(e_S^\top C e_S + \gamma e_S^\top E e_S) \\ =& e_S^\top C e_U + \alpha|S| - (\alpha+1) e_S^\top (C + \gamma E) e_S \end{aligned} \tag{21}$$

Now we define

$$C^+ = C + \gamma E. \tag{22}$$

According to the definition of γ, we have that $\gamma \geq 0$ and elements in C^+ are all non-negative. Thus, $-(\alpha+1)e_S^\top(C+\gamma E)e_S$ is submodular due to the submodularity of negative quadratic function [2]. In addition, we can see that $e_S^\top C e_U + \alpha|S|$ is modular. Thus, $f_c(S) - \gamma(1+\alpha)|S|^2$ is submodular. □

THEOREM 2. *When $\mu \geq \gamma$, the objective function in (17) is submodular and the DLS algorithm has a 1/3-approximation bound in the worst case.*

PROOF. According to Lemma 1, (17) can be rewritten as

$$\sum_{i\in S}\sum_{j\in U-S} c^+_{i,j} - \alpha \sum_{i,j\in S, i\neq j} c^+_{i,j} + \beta \sum_{i\in S} a_i - (\mu-\gamma)|S|^2, \tag{23}$$

where $c^+ i, j$ are elements in C^+ defined in (22). Since $c^+_{i,j} \geq 0$, we see that (23) has the same form as defined in Theorem 1 if $\mu - \gamma \geq 0$. Thus, it is submodular and has a 1/3-approximation bound [9]. □

In practice, we can first compute all the $c_{i,j}$ and γ, and then choose a μ value such that $\mu \geq \gamma$. Therefore, we can always guarantee the submodularity and optimization bound.

3.4 Sparse learning algorithm

Since the dimensionality of subspaces should be relatively small to avoid the curse of dimensionality, in this paper we have a restriction that $|A_m| \leq 2$. So there are $M = O(D(D+1)/2)$ potential $\phi_m(\mathbf{x})$ including 1-dimensional and 2-dimensional subspaces. After all those $\phi_m(\mathbf{x})$ are computed by smoothed histogram estimation and filtered by submodular subspace selection, we need to learn the weight vector $\mathbf{w} = (w_1, \cdots, w_M)$[5] in (11). We require strong sparsity on $\mathbf{w}$, which is not satisfied by the learning framework in (2).

One simple alternative is to replace the ℓ_2 regularization with ℓ_1 regularization [11] which promotes sparsity. However, the FFD model has a nice structure and can offer an even better solution. Let us rewrite (10) as

$$\phi_m(\mathbf{x}) = \ln \frac{p(y=1|[\mathbf{x}]_{A_m})}{1-p(y=1|[\mathbf{x}]_{A_m})} = g\left(p(y=1|[\mathbf{x}]_{A_m})\right) \tag{24}$$

where $g(z) = \ln \frac{z}{1-z}$ is a logit function. We can observe that $\phi_m(\mathbf{x})$ is an increasing function of $p(y=1|[\mathbf{x}]_{A_m})$. In addition, $\phi_m(\mathbf{x}) \geq 0$ if $p(y=1|[\mathbf{x}]_{A_m}) \geq 0.5$ and less than 0 otherwise. Given the monotonic relation between $p(y=1|\mathbf{x})$ and $\phi_m(\mathbf{x})$ in (11), the weight w_m on $\phi_m(\mathbf{x})$ is supposed to be non-negative under the assumption that $p(y=1|[\mathbf{x}]_{A_m})$ estimated by histogram is a weak learner. Given that $\mathbf{w}$ is non-negative, we have that $\|\mathbf{w}\|_1 = \sum_{m=1}^M w_m$. The learning formulation with ℓ_1 regularization becomes:

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & \frac{1}{N}\sum_{i=1}^N \left(1 + e^{-y_i(\mathbf{w}\Phi(\mathbf{x}_i))}\right) + \lambda \sum_{m=1}^M w_m \\ \text{subject to} \quad & \mathbf{w} \geq 0 \end{aligned} \tag{25}$$

Compared with other ℓ_1 norm method, the ℓ_1 norm in our framework is differentiable for all feasible $\mathbf{w}$. Such a smooth optimization problem with simple bound constraints can be efficiently solved by gradient-descent solvers.

Using (25), FFD also enforces more sparsity than traditional ℓ_1 regularization. To see that, assume that the gradient of a positive w_m is negative. When doing gradient descent on it, w_m will tend to decrease its value. But due to the non-negative constraint, it cannot go below 0 and will thus end up at 0, which leads to even more sparse solution than ℓ_1 regularization.

The overall learning algorithm for FFD is shown in Algorithm 1.

Algorithm 1 The learning algorithm for FFD

```
for m = 1 to M do                                  ▷ Histogram estimation
    Build grid G_{A_m}.
    for i = 1 to N do
        Assign x_i to the grid cell indexed by (9)
    end for
    for B ∈ G_{A_m} do
        Compute p(y|B) by (12)
    end for
end for
for i = 1 to N do                                  ▷ Computing φ_m(x_i)
    for m = 1 to M do
        Compute φ_m(x_i) by (10)
    end for
end for
Select subspaces by solving (17) using the DLS algorithm
Learn w by solving (25)
```

3.5 Discussions

We make a few further comments about FFD.

Cross validation. Before executing the complete algorithm of FFD as stated in Algorithm 1, all the hyperparameters should be pre-specified including λ in (25) and (α, β, μ) in (17). Experts can set these hyper-parameters by good intuition. For non-experts, a typical way to tune them is utilizing a k-fold cross-validation where grid search is performed on all hyper parameters to minimize the validation error. However, this brute-force algorithm has a high computational cost. In addition, it does not make use of the validation errors during the grid search.

As a better solution,we use a recent Bayesian optimization technique [28] to tune the hyper-parameters in order to globally minimize the validation error. Specifically, the validation error is modeled as a sample from a Gaussian Process (GP). Each time, we sample the hyper-parameters $(\lambda, \alpha, \beta, \mu)$ which minimize the expectation of its validation error under the assumption of GP. Then, Algorithm 1 is run on this hyper-parameter sample to evaluate its actual cross-validation error. This sample with its validation error

[5]Now M is the number of ϕ_m after submodular subspace selection.

are then incorporated in the previous GP and forms a new GP. This process keeps running until a maximum iteration limit is reached. The best hyper-parameters are the one that have the minimum cross-validation error among all the samples. It has been shown that this Bayesian optimization method for cross validation largely outperforms normal grid search [28].

Usefulness of submodular subspace selection. Though ℓ_1 regularization encourages sparsity, the performance of linear methods with ℓ_1 regularization (including logistic regression) suffers if the input variables are highly correlated or even co-linear [11, 14]. Thus, before training FFD by solving (25), it is necessary and important to do subset selection on all ϕ_m in order to reduce their correlation.

Interpretability. Similar to LR, FFD can provide probability of its prediction in addition to the predicted label. However, in LR model the weights on different features may not be directly comparable. For example, the measurement of blood pressure has a vastly different scale than that of height, and their weights are not comparable. In contrast, the weights $\mathbf{w}$ in FFD do not suffer from this problem because all ϕ_m are on the same scale as they model conditional probabilities. In addition, each $\phi_m(\mathbf{x})$ is an indicator of $p(y = 1|[\mathbf{x}]_{A_m})$ because they have a monotonic relationship as shown in (10). With the sparsity of FFD, we can also identify which feature bags A_m are in fact *sufficient* to make accurate predictions based on the weights. Since kernel density estimation for the subspaces is highly nonlinear and approximates the true conditional distribution, FFD reliably learns the feature interactions.

4. RELATED WORK

Our FFD model is related to density-based logistic regression (DLR) [6] which transforms the original raw features to a new feature representation by KDE and then trains a logistic regression model on the new feature vectors. However, it does not promote sparsity and is not capable of handling feature interaction. In addition, compared to DLR, the feature transformation in FFD allows much faster computation, reducing the training time from $\mathrm{O}(DN^2)$ to $\mathrm{O}(DN)$ and the testing time from $\mathrm{O}(DN)$ to $O(D)$.

In general, FFD is related to recent works on learning linear models with explicit nonlinear feature transformations. For example, random kitchen sinks (RKS) [24] transforms each data $\mathbf{x}$ into a finite-dimensional vector $\phi(\mathbf{x}) \in \mathcal{R}^M$ to approximate the RBF-kernel function for any two inputs $\mathbf{x}$ and $\mathbf{y}$ in such way that $\phi(\mathbf{x})^T\phi(\mathbf{y}) \approx K(x, y)$, where $K(\mathbf{x}, \mathbf{y})$ is the RBF kernel function. This allows highly scalable *linear* classifiers in the transformed space to learn approximately the same decision boundaries as SVM with a RBF kernel in the original input space. The recent fastfood algorithm [17] further speeds up RKS using matrix approximation techniques and reduces the time and space complexities. Other feature mapping techniques include those based on random projection [1, 15, 18, 23], polynomial approximation [21], and hashing [19, 32]

Existing feature mapping techniques, when combined with linear classifiers, can achieve both nonlinear separability and higher scalability of linear classifiers. However, they cannot take advantage of the interpretability of linear classifiers. The feature mapping techniques such as Fourier transformation, random projection, and hashing are all defined in a different space than that of the original features. As a result, these models do not provide a weight for each original feature dimension and cannot offer a clear notion of interpretability. Moreover, all these works except for [18] do not explicitly support sparsity.

Table 1: Comparison of the characteristics of different classifiers.

	LR	SVM-rbf	DLR	RKS	FFD
Interpretable	Yes	No	Yes	No	Yes
Efficient	Yes	No	No	Yes	Yes
Nonlinear	No	Yes	Yes	Yes	Yes
Sparse	Yes	No	No	No	Yes

5. EXPERIMENTAL RESULTS

In this section, we conduct extensive experiments to evaluate the proposed FFD model. We evaluate three versions of FFD.

- FFD-1: FFD with only 1-dimensional subspaces, i.e. $|A_m| \leq 1$, and without submodular subspace selection.
- FFD-2: FFD with 1-dimensional and 2-dimensional subspaces, i.e. $|A_m| \leq 2$, and without submodular subspace selection.
- FFD-sfo: FFD-2 with submodular optimization for subspace selection.

In FFD-2 and FFD-sfo, for ϕ_m involving two dimensions, we picks the top $3D$ out of $D(D-1)/2$ such ϕ_m's according to their accuracies for histogram estimation, where D is the dimensionality of the training samples. For all the three methods, we set the number of bins for numerical features to 50, i.e. $b_d = 50$. For submodular maximization in (17), we use the *sfo* toolbox developed by Andreas Krause [16]. To solve the sparse learning problem in (25), we use the minConf_TMP function[6]. This function is superior at optimizing smooth objective functions subject to bound constraints, which is exactly our case.

Baseline methods. We also consider four other methods for comparison. 1) Logistic regression with ℓ_1 regularization (LR) [11], implemented by Mark Schmidt [25]. 2) Support vector machines with the RBF kernel (SVM-rbf). We use the LibSVM library [5]. 3) Density-based Logistic Regression (DLR) [6], a linear classifier with nonlinear feature mapping based on the naive Bayes assumption. 4) Random Kitchen Sink (RKS) [24], a linear classifier with nonlinear feature mapping that approximates the RBF kernel.

A comparison of the characteristics of these models is given in Table 1. We can see that FFD is the only model that can support *all* of the desirable properties including interpretability, nonlinearity, efficiency, and sparsity.

Cross validation. The hyper-parameters in all methods are tuned via cross validation. The cross validations are performed by Bayesian optimization which has been proved to be far more efficient and accurate than a simple grid search [28]. For the RKS model, we also cross validate all

[6]Available at
`http://www.di.ens.fr/~mschmidt/Software/minConf`

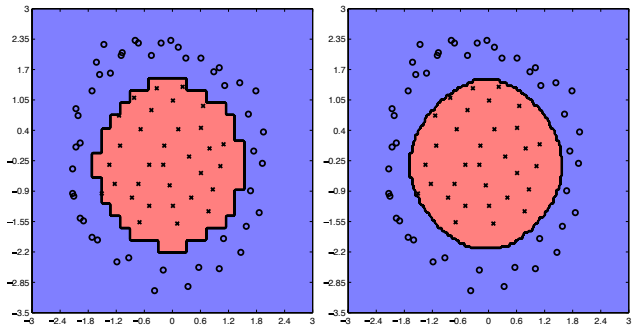

Figure 3: The decision boundary of FFD-1 on a toy example. Left: FFD-1 with 20 bins for each dimension. Right: FFD-1 with 100 bins for each dimension.

types of features including random Fourier features and random binning features.

Visualization of a toy example. We create a nonlinearly separable dataset to visualize the classification ability of FFD method and the effect of using different number of bins, as shown in Figure 3. We can observe that the FFD-1 can perfectly separate the two classes while other linear methods cannot. We also see that, as the number of bins increases, the decision boundary gets smoother.

Comprehensive evaluation. Table 2 shows a comprehensive comparison of all methods on various datasets. All these datasets are publicly available at the UCI repository[7] or the LibSVM website[8]. The datasets are sorted by the number of samples in the datasets. The experiments are run on an off-the-shelve desktop with two 8-core Intel(R) Xeon(R) processors of 2.67 GHz and $128GB$ of RAM. The implementations of all methods are in or through the interface of MATLAB^{TM}. In order for SVM-rbf to run as fast as possible, we set the cache size of LibSVM to $10GB$ which is sufficiently large for all the tested datasets.

From Table 2, we can observe the following facts.

In terms of accuracy, the performance of FFD methods is fairly strong in general. Datasets including checkboard, banana, mnist38 and cod-rna are well-known for its high non-linearity, which can be told by the poor performance of LR on these datasets. However, both FFD-2 and FFD-sfo perform almost as good as SVM-rbf and RKS, which demonstrates their superior nonlinear classification ability. And for datasets including splice and Adult, FFD methods largely outperforms all the other methods.

In terms of running time, we can see that FFD models are very efficient in general, which is fairly comparable with and sometimes even better than the linear machines LR and RKS. For example, the Adult dataset has 14 features, among which 8 are categorical. Models such as LR, SVM-rbf and RKS should first convert the categorical features to numerical features before training. The most popular method, as recommended in [31], is using k binary numerical features to represent an k-category feature. For example, (red,blue,green) can be represented by (1,0,0), (0,1,0) and (0,0,1). We observe that the original 14 features increase to 104 features at the end, which adds to the burden of training. In contrast, our FFD models naturally handle categorical features since each category is a bin in its dimension and no kernel smoothing is needed, leading to superior time efficiency.

We also see that FFD models are much more efficient than nonlinear models such as SVM-rbf and DLR. For the cod-rna dataset with 0.27 million samples, it takes FFD models less than 1 second while SVM-rbf spends over 10 minutes and DLR cannot finish. The improvement is even more salient on the kddcup99 dataset which contains almost 5 million samples. We make kddcup99 a binary classified dataset by setting class 6 to a positive class and other classes to a negative class, since class 6 contains 57% of the samples. It takes FFD models less than one minute while it is prohibitively large for both SVM-rbf and DLR.

However, though FFD-sfo generally has good accuracy and efficiency, we also point out that the cost of submodular optimization is sensitive to the dimensionality of the datasets, and using submodular subspace selection may not be the most sensible for image datasets such as mnist38. However, even for the mnist38 dataset, FFD-sfo is much faster than SVM-rbf and DLR, while FFD-1 and FFD-2 are extremely fast.

In terms of practical usability, FFD models give interpretability and sparsity, while other nonlinear models including SVM-rbf and RKS do not. In summary, FFD is a clear winner considering all the aspects.

Cases of highly correlated features. As stated in Section 3, linear methods including models with ℓ_1 regularization become more unstable and less accurate when highly correlated features or co-linearity exist. Although FFD methods are capable of classifying nonlinear datasets, they are linear models on ϕ_m and thus may also suffer from this issue. In this case, FFD-sfo is better than FFD-2 since it rules out highly correlated ϕ_m by maximizing the submodular objective function in (17). This effect has been partially reflected in Table 2.

To demonstrate the advantage of FFD-sfo over FFD-2 in a more observable way, we conduct another experiment in the feature selection context. First, we make a new splice data set by duplicating each feature, which makes them co-linear. Then, we run FFD-2 and FFD-sfo on both the original and new datasets. For reference, we also run the LR and Lasso [11] models which are state-of-the-art feature selection methods. For FFD methods, one feature is considered selected if it is used by any ϕ_m whose $w_m \neq 0$. Figure 4 shows the curves about accuracy versus number of selected features. First, we observe that FFD methods is much better than LR and Lasso since FFD offers nonlinear separability. Second, although FFD-2 and FFD-sfo have comparable performance on the original dataset, FFD-sfo is way better than FFD-2 on the new dataset, demonstrating the ability of FFD-sfo in addressing the issue of highly correlated or co-linear features.

Real-world clinical prediction. In addition to evaluation on public benchmark datasets, we also test FFD models on a real-world clinical application. This is a collaboration with Barnes-Jewish Hospital, one of the largest hospitals in the US. The task is to predict potential ICU transfers for hospitalized patients based on 34 vital signs. The data col-

[7] `https://archive.ics.uci.edu/ml/datasets.html`
[8] `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

Table 2: Performance of various methods on various datasets. Note that the FFD models have the additional advantages of interpretability and sparsity, which are not found in SVM-rbf and RKS. "N/A" means unfinished in 2 hours or memory overflow.

Dataset	N	D	Performance	LR	SVM-rbf	DLR	RKS	FFD-1	FFD-2	FFD-sfo
breast	683	9	accuracy(%)	96.63	96.05	97.36	97.07	96.34	97.51	97.07
			time(sec)	0.0164	0.0165	0.0574	0.0573	0.0278	0.4242	0.5779
splice	1000	60	accuracy(%)	80.70	87.20	92.00	90.10	91.60	94.60	94.50
			time(sec)	0.0144	0.2468	0.4069	0.0679	0.0835	1.235	2.2159
checkboard	2000	2	accuracy(%)	49.70	93.95	51.00	92.50	51.25	92.20	92.30
			time(sec)	0.0143	0.1454	0.0589	0.8763	0.0188	0.0434	0.07
banana	5300	2	accuracy(%)	56.70	90.45	71.89	89.57	71.74	89.72	89.91
			time(sec)	0.0126	0.6449	0.2354	0.5021	0.0326	0.0516	0.0298
musk	6598	166	accuracy(%)	94.33	90.69	95.21	96.42	96.88	95.06	94.71
			time(sec)	1.2882	58.6402	23.1885	0.7666	1.0141	4.1618	25.139
mnist38	11982	784	accuracy(%)	96.77	99.11	97.09	97.28	96.32	98.69	98.98
			time(sec)	0.9187	230.1897	1621	0.4323	2.0696	1.0952	122.419
magic04	14226	10	accuracy(%)	79.50	86.38	82.59	84.57	81.40	82.21	84.98
			time(sec)	0.2639	20.145	21.6976	1.6225	0.3211	0.188	0.4033
Adult	30162	14	accuracy(%)	84.71	80.43	85.41	83.96	85.40	85.31	85.42
			time(sec)	6.8223	492.7818	81.8001	3.8515	0.4057	3.1718	1.1399
cod-rna	271617	8	accuracy(%)	75.08	91.75	N/A	92.29	77.34	89.53	89.73
			time(sec)	0.785	629.759	N/A	5.7484	0.6583	1.0074	0.7651
kddcup99	4898431	41	accuracy	99.99	N/A	N/A	99.92	99.99	99.99	99.99
			runtime(sec)	179.2541	N/A	N/A	41.3827	46.2968	50.0306	54.0227

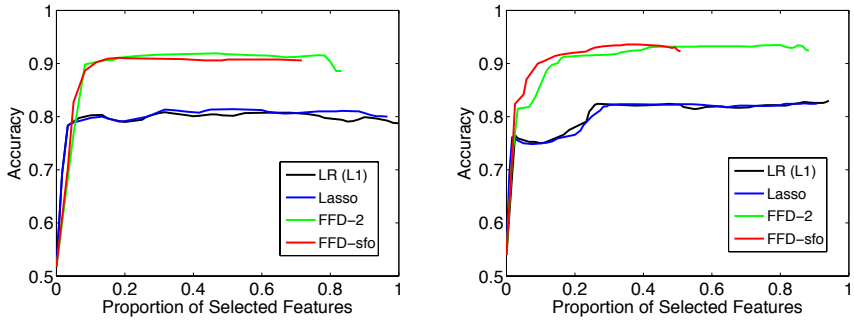

Figure 4: Accuracy vs. proportion of selected features. Left: The original splice dataset; Right: The new splice dataset with co-linear features.

lection process can be found in [6]. On this dataset, FFD-sfo and FFD-2 have 95.37% and 94.59% accuracy, respectively. The results outperform the DLR model with a 93.26% accuracy which is found to be better than LR and SVM [6]. Moreover, FFD is the only model that can offer sparsity and interpretability. This is vitally important in clinical practice since healthcare personnel can be informed of the most important risk factors and take proper actions for prevention and intervention.

6. CONCLUSIONS

Many applications in the big data era find existing classifiers inadequate. They often require not only high accuracy but also high efficiency, sparsity, and interpretability. To date no classifier, linear or nonlinear, excel at all these aspects. We have presented a novel Fast Flux Discriminant (FFD) model which delivers all these desirable properties. FFD learns a linear discriminant on top of a non-parametric feature mapping which captures nonlinear feature interactions. FFD addresses the curse of dimensionality by decomposing the kernel density estimation in the original feature space into low-dimensional subspaces. We have also proposed a submodular optimization framework to select subspaces and to address the problem of highly correlated features and collinearity. Moreover, FFD attains feature sparsity using ℓ_1 regularization. A nice feature of FFD is that its density-based features naturally have non-negative weights and hence allows for smooth optimization, which is not previously possible for ℓ_1 regularization. Empirical results have shown that FFD delivers similar accuracy as state-of-the-art nonlinear classifiers, but with sparsity, interpretability, and much better scalability. To the best of our knowledge, this is the first classifier that possesses all these merits. Given its unprecedented combination of advantages, we believe FFD will become a popular general-purpose classification model

for a large scope of real-world applications such as biomedical prediction.

7. ACKNOWLEDGMENTS

WC and YC are supported in part by the CNS-1017701, CCF-1215302, and IIS-1343896 grants from the National Science Foundation of the United States, a Microsoft Research New Faculty Fellowship, a Washington University URSA grant, and a Barnes-Jewish Hospital Foundation grant. KQW is supported by NSF grants 1149882, 1137211.

8. REFERENCES

[1] D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Science*, 66:671–687, 2003.

[2] F. Bach. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends in Machine Learning*, 2013.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[6] W. Chen, Y. Chen, Y. Mao, and B. Guo. Density-based logistic regression. In *Proceedings of the 19th ACM SIGKDD*, KDD '13, pages 140–148, New York, NY, USA, 2013. ACM.

[7] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[8] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[9] U. Feige and V. S. Mirrokni. Maximizing non-monotone submodular functions. In *In Proceedings of 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS*, page 2007, 2007.

[10] S. Fujishige. *Submodular Functions and Optimization: Second Edition*. Annals of Discrete Mathematics. Elsevier Science, 2005.

[11] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[12] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[13] R. Iyer, S. Jegelka, and J. A. Bilmes. Fast semidifferential-based submodular function optimization. In *International Conference on Machine Learning (ICML)*, Atlanta, Georgia, 2013.

[14] G. James, T. Hastie, D. Witten, and R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer, 2013.

[15] P. Kar and H. Karnick. Random feature maps for dot product kernels. In *Proc. AISTATS*, 2012.

[16] A. Krause. Sfo: A toolbox for submodular function optimization. *The Journal of Machine Learning Research*, 11:1141–1144, 2010.

[17] Q. Le, T. Sarlos, and A. Smola. Fastfood - computing hilbert space expansions in loglinear time. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 244–252, May 2013.

[18] P. Li, T. J. Hastie, and K. W. Church. Very sparse random preojections. In *Proc. SIGKDD*, pages 287–296, 2006.

[19] P. Li and A. Konig. b-bit minwise hasing. In *Proc. WWW*, 2010.

[20] H. Lin and J. Bilmes. Multi-document summarization via budgeted maximization of submodular functions. ACL-HLT '10, pages 912–920, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[21] K. Lin and M. Chen. Efficient kernel approximation for large-scale support vector machine classification. In *Proc. SIGKDD*, 2011.

[22] O. Pele, B. Taskar, A. Globerson, and M. Werman. The pairwise piecewise-linear embedding for efficient non-linear classification. In *Proceedings of The 30th International Conference on Machine Learning*, pages 205–213, 2013.

[23] N. Pham and R. Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *Proc. SIGKDD*, 2013.

[24] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

[25] M. Schmidt, G. Fung, and R. Rosales. Fast optimization methods for l1 regularization: A comparative study and two new approaches. In *ECML*, volume 4701, pages 286–297. 2007.

[26] B. Schölkopf and A. J. Smola. *Learning with kernels*. The MIT Press, 2002.

[27] B. W. Silverman and P. J. Green. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[28] J. Snoek, H. Larochelle, and R. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pages 2960–2968. 2012.

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

[30] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(3):480–492, 2012.

[31] C. wei Hsu, C. chung Chang, and C. jen Lin. A practical guide to support vector classification, 2010.

[32] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proc. ICML*, 2009.

[33] G. Yuan, C. Ho, and C. Lin. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100:2584–2603, 2012.