Prediction-time Efficient Classification Using Feature Computational Dependencies

Liang Zhao, Amir Alipour-Fanid, Martin Slawski, Kai Zeng
George Mason University
{lzhao9,aalipour,mslawsk3,kzeng2}@gmu.edu

ABSTRACT

As machine learning methods are utilized in more and more real-world applications involving constraints on computational budgets, the systematic integration of such constraints into the process of model selection and model optimization is required to an increasing extent. A specific computational resource in this regard is the time needed for evaluating predictions on test instances. There is meanwhile a substantial body of work concerned with the joint optimization of accuracy and test-time efficiency by considering the time costs of feature generation and model prediction. During the feature generation process, significant redundant computations across different features occur in many applications. Although the elimination of such redundancies would reduce the time cost substantially, there has been little research in this area due to substantial technical challenges involved, especially: 1) the lack of an effective formulation for feature computation dependency; and 2) the nonconvex and discrete nature of the optimization over feature computation dependency. In order to address these problems, this paper first proposes a heterogeneous hypergraph to represent the feature computation dependency, after which a framework is proposed that jointly optimizes the accuracy and the exact test-time cost based on a given feature computational dependency. A continuous tight approximation to this original problem is proposed based on a non-monotone nonconvex regularization term. Finally, an effective nonconvex optimization algorithm is proposed to solve the problem, along with a theoretical analysis of the convergence conditions. Extensive experiments on eight synthetic datasets and six real-world datasets demonstrate the proposed models' outstanding performance in terms of both accuracy and prediction-time cost.

CCS CONCEPTS

• **Computing methodologies → Cost-sensitive learning**;

ACM Reference Format:
Liang Zhao, Amir Alipour-Fanid, Martin Slawski, Kai Zeng. 2018. Prediction-time Efficient Classification Using Feature Computational Dependencies. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220117

Table 1: An example of the features extracted from a set of raw data. N denotes the number of elements in input feature vector x.

Feature ID:Name	Description	Generation Time
V_1: mean	$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x(i)$	0.672 microsecond
V_2: median	The higher half value of a data sample.	4.365 microsecond
V_3: MAD[1]	$MAD = median(\lvert x(i) - median(x)\rvert)$	8.346 microsecond
V_4: STD[1]	$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x(i) - mean(x))^2}$	1.608 microsecond
V_5: Skewness	$\gamma = \frac{1}{N}\sum_{i=1}^{N}(x(i) - mean(x)/\sigma)^3$	14.917 microsecond
V_6: Kurtosis	$\beta = \frac{1}{N}\sum_{i=1}^{N}(x(i) - mean(x)/\sigma)^4$	14.095 microsecond
V_7: MAX	$H = (Max(x(i))\rvert_{i=1...N})$	0.464 microsecond
V_8: MIN	$L = (Min(x(i))\rvert_{i=1...N})$	0.652 microsecond
V_9: Mean Square (ms)	$MS = \frac{1}{N}\sum_{i=1}^{N}(x(i))^2$	1.147 microsecond
V_{10}: Root Mean Square	$RMS = \sqrt{MS(x)}$	1.273 microsecond
V_{11}: Pearson's Skewness	$3 \cdot (mean(x) - median(x))/\sigma$	8.011 microsecond

1 INTRODUCTION

Machine learning methods are now widely used in many real-world applications, including earthquake detection [28], adult content filtering [15], and intruder detection [26]. Additional requirements from industrial fields must be taken into account, especially the timeliness of the prediction [31]. Given that Google expects to process over 40,000 search queries every second on average in 2018, running a machine learning algorithm is clearly impractical unless it is capable of generating timely predictions in tens of milliseconds. Also, as machine learning models are being applied in smaller devices, the requirements in terms of the CPU time and energy consumption are becoming higher and higher [27]. This means that the prediction time cost is a hurdle machine learning must overcome if it is to be widely adopted outside academic settings. The feature extraction cost dominates the test-time runtime cost, especially when linear models such as those commonly used in industrial settings are utilized [3, 6, 9, 14, 22].

To address this problem, in recent years there has been a steady increase in the amount of research on test-time cost-aware machine learning. This research can generally be categorized into implicit and explicit methods. Implicit methods typically employ boosting, heuristic, or greedy strategies to guide the model towards greater test-time efficiency [17, 19, 20]. Explicit methods optimize criteria involving a trade-off between prediction accuracy and prediction cost [8, 10, 12]. A key observation when minimizing test-time cost is that the costs for extracting different feature subsets vary. For example, for intruder detection, many features can be extracted from a raw input time series, including mean and standard deviation. Naturally, the time cost for extracting standard deviation is larger than that required to calculate the mean. Existing methods generally aim at the selection of those features with relatively low cost while still achieving high prediction accuracy.

In a number of important applications, some computations for generating different features tend to be shared [30], whereupon we refer to their generation processes as being ***computationally***

[1]MAD: median absolute deviation; STD: standard deviation

dependent on each other. Table 1 illustrates some of the features commonly used in signal processing for intruder device detection. As shown in the table, the features "mean", "Kurtosis", "Skewness", "standard deviation", and "Pearson's Skewness" all share the computation of "mean"; the feature "MAD" also contains the computation of "median"; and the computation for the feature "root mean square" includes that of "mean square". Therefore, in our machine learning model, if both "mean" and "standard deviation" are selected then it is only necessary to calculate "mean" once. Similarly, if "mean square" has already been selected, then the additional cost of adding the feature "root mean square" requires only the calculation of the root of a scalar value, which considerably reduces the extra time cost incurred when including that additional feature.

Even though the apparent potential for cost reduction resulting from the aforementioned ***feature computational dependency*** deserves a comprehensive consideration and treatment, as yet little attention has been paid to this issue due to several technical challenges. **1) Difficulty in optimizing the cost associated with computationally dependent feature sets**. To identify a globally optimal set of features, a suitable criterion for quantifying the costs of all possible feature set is required. Unlike the situation in existing work, in the formulation proposed herein the total time cost of all the features combined will no longer be the direct summation of the respective costs for individual features. Instead, each candidate set of features will have its exclusive set of shared computations with the corresponding shared time cost. Given that there are exponentially many possible feature subsets, enumeration of the individual costs for each of possible set is prohibitive. **2) Ineffectiveness of convex approximation for cost optimization with feature dependency**. In addition to feature selection, which is well known as a computationally hard problem, shared computations of the selected features should be counted once in cost optimization, which is actually a second discrete problem in which positive integers are mapped to {0,1}. This requirement typically cannot be satisfied by a convex approximation such as the ℓ_p norm ($p \in [1, 2)$) as shown in Section 4.3.1 below. **3) Algorithmic efficiency for a re-weighted nonconvex-regularized problem**. A complex optimization problem with exponentially many solutions requires efficient methods. Moreover, to ensure that the model is applicable in real-world applications with large datasets optimization methods that are efficient (ideally with linear complexity) and scalable are preferred.

To the best of our knowledge, there is no existing work capable of addressing all the above challenges and providing a concrete framework for the formulation and treatment of cost-efficient machine learning with feature computation dependency. In order to address these challenges simultaneously, this paper proposes a comprehensive framework that explicitly minimizes the prediction error and runtime cost by optimally utilizing the feature computational dependency among different features. The feature computational dependency is represented in terms of a heterogeneous hypergraph whose nodes are the features and the edges are the computation components of the features; the feature computational dependency is then embedded into the formulated framework. The resulting optimization problem is nonconvex and discontinuous, which is difficult to solve. To address this, we propose a tight relaxation equivalent to the original problem, which is easier and more efficient to solve. A new efficient algorithm based on nonconvex Alternating Direction Methods of Multipliers (ADMM) is proposed which has lately been shown to converge to local optima under specific conditions. The major contributions of this paper can be summarized as follows:

- We characterize feature computational dependency using heterogeneous hypergraphs that yield a concise representation of feature dependency and the computational costs associated with each feature subset.
- We optimize prediction accuracy and test-time efficiency utilizing feature computational dependency. A continuous relaxation of the original discrete function for the runtime cost is proposed to provide a tight approximation. This approximation is proved to be equivalent to a class of re-weighted nonconvex regularized problems.
- We develop an effective algorithm to solve these nonconvex problems with theoretical guarantees. The proposed algorithm is fast and scalable to large datasets. A theoretical analysis of the convergence properties is provided as well.
- We conduct extensive experiments on eight synthetic datasets and six real-world datasets. The proposed method is compared with several state-of-the art methods. The analyses of the performance and effectiveness of feature selection demonstrate the advantage the proposed methods over existing methods.

2 RELATED WORK

At the beginning of this section, we briefly review prior work on test-time cost-efficient models, including both implicit and explicit methods. We then continue with an overview on related areas including budgeted learning, cost-aware data acquisition, and sparse feature learning.

Implicit cost-aware methods. There is typically a trade-off between prediction accuracy and prediction cost when incorporating runtime into model optimization. Implicit cost-efficient methods do not necessarily model this trade-off directly, but tend to employ heuristic or greedy strategies to guide the model prediction towards cost efficiency. There is extensive research under this category, including: 1) *Cascades of classifiers.* Here, several classifiers are ordered as a sequence of stages. Each classifier can either reject inputs by predicting them, or pass them on to the next stage. To reduce the test-time cost, these cascade algorithms enforce that classifiers in early stages use very few and/or cheap features and reject many easily classifiable inputs [17, 18]. 2) *Decision-tree based.* Decision tree (and forest) induction methods have been extensively used for decision making when the cost of acquiring the features are considered [13]. For example, Tan and Schlimmer [20] employed an entropy-based strategy to estimate the cost while Ferri et al. [7] leveraged the feature cost to prune the tree after it had been built. Li et al. [14] employed a cost-efficient decision tree based on a heuristic strategy to coarsely partition the feature space, and then applied local SVM classifiers to further refine them. 3) Boosting-based. For example, Reyzin et al. [19] extended AdaBoost and employed weak learners with fewer features in order to reduce the feature cost.

Explicit cost-aware methods. In general, these methods explicitly aim at a balance between prediction accuracy and cost, for example by jointly optimizing a trade-off or optimizing the accuracy under the constraint of a specified cost budget. For this category, the most common method is ℓ_1-regularization, where a sparse set of features will be learned in order to not only ensure model generalization but also a reduction in computational total cost [5]. To consider the different costs of the various features, a number of approaches have been proposed. For example, Grubb et al. [8] proposed an algorithm for "anytime prediction" which outputs predictions with increasing quality as the cost budget increases. Xu et al. [24] developed Greedy Miser, a variant of regular stage-wise regression, which updates the selected features using a greedy optimization strategy. Kusner et al. [12] formulated the cost-sensitive feature selection as an approximate submodular optimization problem, while Huang and Wang [10] developed a genetic algorithm-based method to maximize an objective function consisting of classification accuracy and inverse cost.

Budget learning and cost-aware data acquisition. In addition to the test-time cost, which is the focus of this paper, several related works pay close attention to the training cost, including budgeted learning and cost-aware data acquisition. The difference between active feature acquisition and budgeted learning is that budgeted learning usually has a hard budget set up-front, while active feature acquisition does not have a hard budget [11]. For example, Deng et al. [4] designed algorithms for the multi-armed bandit problem to select specific features for specific instances under a limited budget. Nan and Saligrama [16] developed an adaptive method which learns both a low-cost and a high-cost models by maximizing the utilization of low-cost models while maintaining the performance, and hence controlling total cost. Cost-aware data acquisition is commonly applied in models for medical diagnosis. For example, Ling et al. [15] proposed a lazy-tree learning to jointly minimize the misclassification cost and the sum of feature costs.

However, none of the above methods considers computation dependency and thus do not factor in redundant computations among features to further reduce their computational cost. To address this problem, this paper proposes an optimization problem based on the representation of feature computation dependency in terms of heterogeneous hypergraphs and proposes an effective algorithm to select those features with a low total cost.

3 PROBLEM SETUP

Define $X = \{X_1, X_2, \cdots, X_n\} \in \mathbb{R}^{n\times k}$ as the input data containing n samples under k features, where each sample is a row vector $X_j \in \mathbb{R}^{1\times k}$. The i-th element of X_j corresponds to a feature $V_i \in V$, where $V = \{V_1, \cdots, V_k\}$. For each X_j, there is a corresponding $Y_j \in \{0, 1\}$ such that $Y_j = 1$ means it is labeled as positive; $Y_j = 0$ otherwise. The prediction runtime consists of: 1) feature generation and 2) model prediction. The prediction runtime depends on how many and which features are to be selected and can be denoted as $\mathcal{T} = \mathcal{T}_1 + \mathcal{T}_2 + const$, which is the sum of the parts that are relevant and irrelevant to the selected features. Specifically, "*const*" denotes the runtime that is irrelevant for the selected features, such as the computation of the sigmoid function when using logistic regression for prediction, given the already calculated linear combination of all the feature values. Moreover, $\mathcal{T}_1$ denotes the time for feature generation and $\mathcal{T}_2$ represents the time for feature utilization, namely the computation directly utilizing the generated features (e.g., the first layer of neural network). For the latter, the computation time is only relevant to the number of features selected, while for the former, the computation time is not only relevant to how many but also which features are selected. For example, Table 1 shows the feature generation runtime for 11 features.

As shown in Table 1, different features potentially share a number of computations during their generations in which case we say that these features have *feature computational dependency*. When evaluating the computational cost of a group of features, it is desirable that only distinct computations are counted. To explicitly express the selected features U and their time cost estimation $\mathcal{T}(U;\mathcal{G})$ based on the feature computational dependency $\mathcal{G}$, the entire computational runtime can be rewritten as $\mathcal{T}(U;\mathcal{G}) = \mathcal{T}_1(U;\mathcal{G}) + \mathcal{T}_2(U) + const$, where the feature computation dependency $\mathcal{G}$ is used to take into account shared computations for all possible feature subsets.

At a high level, our goal is to select a subset of features $U \subseteq V$ that can jointly achieve fast and accurate prediction. One problem is to maximize the prediction accuracy within the required prediction time, which can be formulated as the minimization classification error subject to an upper bound on the total prediction time:

$$\min_{W, U\subseteq V} \mathcal{L}(Y, f(W, X)), \quad s.t.\ \mathcal{T}(U;\mathcal{G}) \leq \tau \tag{1}$$

where τ is an upper bound on the admissible prediction time. Moreover, $W \in \mathbb{R}^{1\times k}$ is the set of feature weights such that W_i denotes the weight for feature V_i; $\mathcal{L}(\cdot)$ is the empirical loss function quantifying prediction error, such as the logistic loss or hinge loss for classification problems. Finally, $f(W, X)$ denotes the corresponding classifier. Alternatively, when there is no explicit upper bound on prediction time, the prediction error and time cost can be jointly minimized:

$$\min_{W, U\subseteq V} \mathcal{L}(Y, f(W, X)) + \lambda\mathcal{T}(U;\mathcal{G}) \tag{2}$$

where $\lambda \geq 0$ is the trade-off parameter between classification error and time cost.

Solving the above problems in Equations (1) and (2) entail two challenges: 1. An exponentially large number of records in $\mathcal{G}$ for shared computations. Due to the existence of feature computational dependency, each combination of features will have its own exclusive pattern of shared computations. However, it is not feasible to enumerate all possible feature subsets and the associated computational costs. A concise representation of $\mathcal{G}$ is mandatory as a first step towards efficient optimization. 2. The joint optimization of continuous and discrete terms. In Equations (1) and (2), optimization for W is a continuous problem while that in U is discrete.

4 MODELS

To solve the problems in Equations (1) and (2) and to address the above challenges, we first propose a new heterogeneous hypergraph for modeling the feature computational dependency (FCD) and then propose our model named Cost-Aware classification using the FCD Heterogeneous hypergraph (CAFH).

4.1 Heterogeneous hypergraph for feature computational dependency

This section presents a concise representation of the feature computational dependency $\mathcal{G}$ via a heterogeneous hypergraph.

For each feature combination, in order to specify shared computations as well as those exclusive to each feature, the concept of "feature computation component (FCC)" is employed. FCCs are

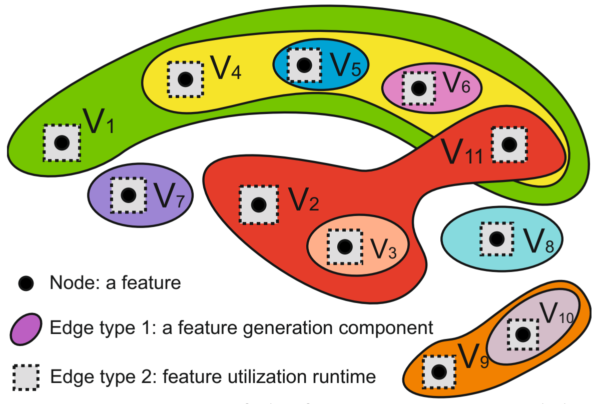
V1
V4
V5
V6
V11
V7
V2
V3
V8
V10
V9
Node: a feature
Edge type 1: a feature generation component
Edge type 2: feature utilization runtime

Figure 1: An overview of the feature computational dependency heterogeneous hypergraph for Table 1.

the basic units that collectively represent the computation process underlying the generation of all features. For example, standard deviation would have three FCCs, the first being the computation of "mean", the second being the calculation of the "standard deviation" using the computed mean, and the third being the computation where the prediction model utilizes the computed standard deviation to make its prediction. In this example, the first two FCCs are for feature generation while the third is for the feature utilization by the prediction model.

This means that each feature can contain multiple FCCs and an FCC can also be shared by multiple features. This notion can be naturally captured by a hypergraph. However, since in our problem there are two types of FCCs, one for feature generation (Type 1) and one for utilization (Type 2), a new heterogeneous hypergraph must be formulated to represent the feature computational dependency $\mathcal{G}$, which is subsequently referred to as **feature computational dependency heterogeneous hypergraph (FCD heterogeneous hypergraph)**. The formal definition is as follows:

Definition 1 (FCD Heterogeneous Hypergraph). *An **FCD heterogeneous hypergraph** is a heterogeneous hypergraph where a node is a feature and an hyperedge is an FCC. There are two types of hyperedges: 1) Type 1: FCCs for feature generation, and 2) Type 2: FCCs for feature utilization. Several nodes can be linked by the same Type-1 hyperedge if they share the same FCC. Each feature has only one Type-2 hyperedge. More formally, denote an FCD heterogeneous hypergraph as $\mathcal{G} = (V, E, w(E))$, where the node set is the set of features V and the hyperedge set E is the set of all the FCCs. $w(E)$ denotes the weights of all the hyperedges. The weight of hyperedge E_i is represented as $w(E_i)$, denoting time cost for the corresponding FCC.*

The FCD heterogeneous hypergraph of the feature set in Table 1 is shown in Figure 1. In this example, there are 11 nodes corresponding to features and 10 hyperedges denoting the corresponding computation components. Each node is linked to at least one hyperedge, while each hyperedge covers at least one node. There are 6 *singleton hyperedges*, each of which cover a single node, signifying that the computation is exclusive to single features. For example, as shown in Table 1 and Figure 1, the features "Skewness" and "standard deviation" both include the computation component "mean" so that these two nodes are linked by the hyperedge of the same computation component.

The proposed FCD heterogeneous hypergraph has several basic properties: 1) The total computation time for a feature is the sum of all the hyperedges having connections to it; 2) The total computation time for a set of features is the sum of all the hyperedges having connections to them; 3) All the Type-2 hyperedges typically have equal weights with each other; and 4) Each Type-2 hyperedge spans one and only one node.

4.2 Cost-aware classification using FCD Heterogeneous Hypergraph (CAFH)

This section presents our model for Cost-aware classification using FCD Heterogeneous Hypergraph (CAFH). The selection of a feature is indicated by its weight: $W_i = 0$ means feature V_i is not being used and thus can be ignored in the prediction phase, and $W_i \neq 0$ means it is included. Accordingly, we use the indicator function: we have $I(W_i) = 0$ when $W_i = 0$, and when $W_i \neq 0$, then $I(W_i) = 1$; these two cases correspond to exclusion and inclusion of feature i, respectively. The set of selected features $U \subseteq V$ is thus defined as $U = \{V_i | I(W_i) = 1, V_i \in V\}$. Then FCCs involved are $E' = \bigcup_v^U e(v) \subseteq E$ and the total runtime is $\mathcal{T}_1(U;\mathcal{G}) + \mathcal{T}_2(U) = \sum_e^{E'} w(e)$.

Define $\mathbf{I}(W) \in \{0,1\}^{|V|\times 1}$ as an indicator vector such that its ith element is $[\mathbf{I}(W)]_i = I(W_i)$. Using matrix representation, the above notion of cost can be concisely expressed as follows:

$$\mathcal{T}_1(U;\mathcal{G}) + \mathcal{T}_2(U) = \sum\nolimits_e^{E'} w(e) = D^T \cdot \mathbf{I}(H \cdot \mathbf{I}(W)), \tag{3}$$

where $H \in \{0,1\}^{|E|\times|V|}$ is the incidence matrix of the proposed FCD heterogeneous hypergraph such that $H_{i,j} = 1$ means the hyperedge $E_i \in E$ is an incident edge of $V_j \in V$; and, $H_{i,j} = 0$ means E_i is not connected to V_j. $D \in \mathbb{R}^{|E|\times 1}$ is a vector whose elements are the weights of the hyperedges in FCD heterogeneous hypergraph, namely $D_i = w(E_i)$ for each ith hyperedge $E_i \in E$. In practice, typically $\mathcal{T}_1 \gg \mathcal{T}_2$ [25], so that in some practical applications we may only need to consider Type 1 edge when formulating H and D.

Considering $\mathcal{T}(U;\mathcal{G}) = \mathcal{T}_1(U;\mathcal{G}) + \mathcal{T}_2(U) + const$, Equations (1) and (2) can be transformed to the following two optimization problems, respectively:

$$\min_W \mathcal{L}(W), \ s.t., \ D^T \cdot \mathbf{I}(H \cdot \mathbf{I}(W)) \leq \tau - const, \tag{4}$$

$$\text{and} \quad \min_W \mathcal{L}(W) + \lambda \cdot D^T \cdot \mathbf{I}(H \cdot \mathbf{I}(W)), \tag{5}$$

where the constant term "const" has been absorbed and the denotation of $\mathcal{L}(Y, f(W,X))$ in Equation (1) has been simplified into $\mathcal{L}(W)$.

4.3 Optimization Objective

The optimization problem in Equations (4) and (5) are highly discrete and nonconvex, which makes optimization a demanding task. In this section, we will transform the original problem in order to solve it effectively with a theoretical guarantee. As the formulations in Equations (4) and (5) are equivalent, we will focus on the regularization form (5) in the following.

First, Equation (5) can be simplified by applying an equivalent form where the inside indicator function is replaced with an element-wise absolute-value operation:

$$\min_W \ \mathcal{L}(W) + \lambda \cdot D^T \cdot \mathbf{I}(H \cdot |W|) \tag{6}$$

where $|W| \in \mathbb{R}^{|V| \times 1}$ is an absolute valued vector of W such that each element $[|W|]_i = |W_i|$.

As next step, the second term in Equation (6) is replaced by a simple indicator function in an auxiliary variable M

$$\min_{W,M} \mathcal{L}(W) + \lambda \cdot D^T \cdot \mathbf{I}(M), \ s.t., \ M = H \cdot |W| \tag{7}$$

which has two nonconvex parts due to the indicator function as well as the nonlinear equality constraint. In the following, we will focus on finding an effective convex relaxation.

4.3.1 Continuous relaxation of the objective function. The discontinuity and nonconvexity of the indicator function makes an effective and efficient optimization difficult. A (re-weighted) ℓ_1-norm is conventionally used as a convex relaxation of the cardinality function (aka ℓ_0-norm). But this convex relaxation is too loose for our problem because it totally discards feature computational dependency. Indeed, relaxing the indicator function in (6) or (7) into the absolute value function, one obtains the optimization problem

$$\min_W \mathcal{L}(W) + \lambda \cdot D^T \cdot |H \cdot |W|| \tag{8}$$

which entails that the FCCs associated with each row of H are weighted by the ℓ_1-norm of W; the penalty thus is composed of individual feature contributions and hence does not capture at all the aspect of cost sharing. In order to take into account feature dependency, we consider a nonconvex regularization term (which is actually concave) that yields a tight continuous approximation to the proposed form of discrete regularization. Specifically, we leverage a re-weighted nonconvex regularization to achieve the approximation.

$$\min_{W,M} \mathcal{L}(W) + \lambda \cdot \mathcal{R}_c(M, D) \ s.t., \ M = H \cdot |W| \tag{9}$$

where $\mathcal{R}_c(M, D)$ denotes a re-weighted version of the nonconvex regularization term such that $\mathcal{R}_c(M, D) = \sum_i^{|E|} D_i \cdot \mathcal{R}(M_i)$. Here $\mathcal{R}(\cdot)$ can be a commonly used concave regularization term such as MCP, SCAD, and ℓ_p quasi-norms $(0 < p < 1)$ [2]. For example, when we use re-weighted ℓ_p quasi-norms, then $\mathcal{R}_c(M, D) = \|\text{diag}(D^{1/p}) \cdot M\|_p^p$, which is easy to compute and also satisfies the triangle inequality. Here $\text{diag}(D^{1/p})$ denotes the diagonal matrix whose diagonal elements are the vector pth root of D, namely $[\text{diag}(D^{1/p})]_{i,i} = D_i^{1/p}$. As a nonsmooth component, we can use proximal algorithms to handle the second term when the proximal operator can be computed in closed form, which is only the case when p is equal to some special values, i.e., $p = 1/2$ or $p = 2/3$.

4.3.2 Convex equivalence of the nonconvex constraint. The constraint $M = H \cdot |W|$ is nonconvex due to the non-linearity of $|W|$. Here we propose a convex constraint and prove their equivalence.

First, we introduce the auxiliary variables $B^+ \in \mathbb{R}^{|V| \times 1}$ and $B^- \in \mathbb{R}^{|V| \times 1}$ such that their ith elements $[B^+]_i = \max(0, W_i)$ and $[B^-]_i = \max(-W_i, 0)$. Therefore, $W = B^+ - B^-$, and we have $|W| = B^+ + B^-$ if given $B_i^+ \cdot B_i^- = 0, i = 1, \cdots, |V|$. Using matrix notation, we can denote $W = \Omega_1 \cdot B$ and $|W| = \Omega_2 \cdot B$, where $B = [B^+; B^-] \in \mathbb{R}^{2|V| \times 1}$, $\Omega_1 = [A, -A]$, and $\Omega_2 = [A, A]$. $A \in \mathbb{R}^{|V| \times |V|}$ denotes the identity matrix. Therefore, Equation (7) is transformed into the following:

$$\min_{M, B \geq 0} \mathcal{L}(\Omega_1 \cdot B) + \lambda \cdot \mathcal{R}_c(M, D), \ s.t., \ M = H \cdot \Omega_2 \cdot B \tag{10}$$

This indicates that the equality constraint in Equation (10) is convex. Now we can formally state and prove the equivalence of Equations (9) and (10):

THEOREM 4.1 (EQUIVALENCE OF FORMULATIONS (10) AND (9)). *When $\mathcal{R}_c$ is strictly monotonically increasing in $[0, +\infty)$, formulations* (10) *and* (9) *are equivalent. When $\mathcal{R}_c$ is monotonically non-decreasing in $[0, +\infty)$, the optimal solution of Equation* (10) *must be the optimal solution of Equation* (9).

PROOF. As defined above, $B = [B^+; B^-]$. The proof amounts to proving $B_i^+ \cdot B_i^- = 0$ for any $i = 1, \cdots, |V|$ for the optimal solution of B in Equation (10). In the following, we prove by contradiction. Assume there exists the situation that for some values of i we have $B_i^+ \cdot B_i^- \neq 0$, this means both $B_i^+ > 0$ and $B_i^- > 0$ for the optimal solution. Now we prove there must exist another solution $B_i^{+'}, B_i^{-'}$ satisfying $B_i^{+'} \cdot B_i^{-'} = 0$ that provides a solution that is better (when $\mathcal{R}_c$ is strictly monotonically increasing) or not worse (when $\mathcal{R}_c$ is monotonically non-decreasing).

Without loss of generality, we assume that for any i, $B_i^+ \geq B_i^-$. By denoting $\Delta B_i = B_i^+ - B_i^- \geq 0$, we denote a new solution B' that replaces B_i^+ and B_i^- in the original solution B with ΔB_i and 0, respectively. We now prove that B' is better than B when $\mathcal{R}_c$ is strictly monotonically increasing and is not worse than B when $\mathcal{R}_c$ is monotonically non-decreasing. First, it is obvious that $\Omega_1 \cdot B \equiv \Omega_1 \cdot B'$. Then, we can denote $H_{\cdot,i}$ as the ith column of H. Provided the fact that $H \geq 0$ and any of its column is not all-zeros, we have $M' = H \cdot \Omega_2 \cdot B' < M = H \cdot \Omega_2 \cdot B$, and thus $\mathcal{R}_c(M', D) < \mathcal{R}_c(M, D)$ when $\mathcal{R}_c$ is strictly monotonically increasing, while $\mathcal{R}_c(M', D) \leq \mathcal{R}_c(M, D)$ when $\mathcal{R}_c$ is monotonically non-decreasing. Therefore, the proof is completed. □

In the domain $[0, +\infty)$, common concave regularizers are indeed monotonically non-decreasing, including SCAD and MCP [2]. The ℓ_p-quasi norm [23] and the Log-Sum Penalty (LSP) [1] are strictly monotonically increasing.

5 PARAMETER OPTIMIZATION

We propose an algorithm based on nonconvex Alternating Direction Methods of Multipliers (ADMM) [23] to achieve the optimization of the model in Equation (1). Here we provide the algorithm when $\mathcal{R}_c(M, D)$ is instantiated by the ℓ_p-quasi norm; the algorithm can easily accommodate other nonconvex regularization terms such as LSP and MCP introduced above by embedding their corresponding proximal operators. We employ the following augmented Lagrangian form of the original problem in Equation (10):

$$L_\rho(M, B) = \mathcal{L}(\Omega_1 B) + \lambda \|\text{diag}(D^{1/p}) M\|_p^p + \frac{\rho}{2} \|M - H\Omega_2 B + \Gamma\|_F^2 \tag{11}$$

where ρ is the penalty parameter and Γ is the dual variable. Thus, solving Equation (11) amounts to alternately optimizing the subproblem of B and that of M, as illustrated in Algorithm 1 and detailed in the following.

1. Update B.

The subproblem of B-update is as follows:

$$\min_{B \geq 0} \mathcal{L}(\Omega_1 \cdot B) + \frac{\rho}{2} \|M - H\Omega_2 B + \Gamma\|_F^2 \tag{12}$$

Algorithm 1 Parameter Optimization Algorithm

```
Require: $\Omega_1, \Omega_2, H, D$, and $\eta$.
Ensure: Solutions of $B$ and $M$.
 1: Initialize $\rho = 1, B, M = \mathbf{0}$.
 2: Choose $\varepsilon_p > 0$ and $\varepsilon_d > 0$.
 3: repeat
 4:     repeat
 5:         $\Delta B = B - \eta \nabla_B L_\rho(M, B)$
 6:         $B \leftarrow \max(\Delta B, \mathbf{0})$
 7:     until Convergence
 8:     for $i = 1, \cdots, |E|$ do
 9:         $M_i \leftarrow$ Equation (15)
10:     end for
11:     Calculate the primal residual $p$ and dual residual $d$.
12:     if $r > 10d$ then
13:         $\rho \leftarrow 2\rho$
14:     else if $10r < d$ then
15:         $\rho \leftarrow \rho/2$
16:     else
17:         $\rho \leftarrow \rho$
18:     end if
19: until $p < \varepsilon^p$ and $d < \varepsilon^d$
```

where we use proximal gradient descent with a backtracking Armijo line search [28] to solve this problem. The following proximal gradient is calculated at each iteration:

$$\text{prox}_{\geq 0}(B - \eta \nabla_B L_\rho(M, B)) = \max(B - \eta \nabla_B L_\rho(M, B), \mathbf{0}) \tag{13}$$

2. Update M.

The subproblem of M-update is as follows:

$$\min_{M \geq 0} \lambda \|\text{diag}(D^{1/p})M\|_p^p + \frac{\rho}{2}\|M - H\Omega_2 B + \Gamma\|_F^2 \tag{14}$$

which is separable and each subproblem of M_i is as follows:

$$\min_{M_i \geq 0} h(M_i) = \lambda D_i \cdot M_i^p + \frac{\rho}{2}(M_i - H_i \Omega_2 B + \Gamma_i)^2 \tag{15}$$

which has analytical solutions when p is equal to special values, namely when $p = 1/2$ or $p = 2/3$:

1. **When** $p = 1/2$. By denoting $M_j^{1/2} = x$, the derivative of Equation (15) can be transformed to the following:

$$x^3 - (H_i \Omega_2 B - \Gamma_i)x + \lambda/(2\rho) D_i = 0 \tag{16}$$

where $x^* = \{x_1, x_2, x_3\} \subset \mathbb{C}$ is the set of analytical solutions to the cubic equation using Cardano's formula [1]. $\mathbb{C}$ denotes the complex value domain. Therefore, the analytical solution to M_i is:

$$M_i^* = \max(\max(x_r^*), 0)^2, \text{where } x_r^* = \{s | s \in \mathbb{R}, s \in x^*\} \tag{17}$$

2. **When** $p = 2/3$. By denoting $M_j^{1/3} = x$, the derivative of Equation (15) can be transformed to the following:

$$x^4 - \rho(H_i \Omega_2 B + \Gamma_i)x + \lambda/(2\rho) D_i = 0 \tag{18}$$

Therefore, we have:

$$M_i^* = \begin{cases} \max(\max(x_r^*), 0)^3, & \text{when } x_r^* \neq \emptyset \\ 0 & , \text{when } x_r^* = \emptyset \end{cases} \tag{19}$$

where $x_r^* = \{s | s \in \mathbb{R}, s \in x^*\}$ is the set of real-number solutions.

Algorithm Initialization. The algorithm is initialized as the case for $p = 1$, which is a convex problem and can provide a good initial guess. In addition, ADMM typically does not support a convergence guarantee for nonconvex problems in general situations. However, we can show that the proposed ADMM-based method is guaranteed to converge to a local minimizer under certain conditions, as described in the following theorem:

Theorem 5.1 (Convergence Analysis). *Given that the loss function $\mathcal{L}(\cdot)$ is a Lipschitz-differentiable function (e.g., logistic loss), and the function $\mathcal{L}(\Omega_1 \cdot B) + \lambda \cdot \mathcal{R}_c(H\Omega_2 B, D)$ is a coercive function [1], then the proposed ADMM-based method will converge when optimizing Equation* (10) *when the following conditions are satisfied: 1) the matrix $H \cdot \Omega_2$ has full rank, 2) the number of FCCs is not larger than the number of features.*

Sketch of Proof. Due to the space limitation, the detailed proof is provided in our supplementary material[1]. The following is the sketch of proof. The sufficient conditions for the convergence of a nonconvex ADMM is provided in Theorem 1 of [23], where five assumptions must be satisfied. In our method, because of the coercivity of $\mathcal{L}(\Omega_1 \cdot B) + \lambda \cdot \mathcal{R}_c(H\Omega_2 B, D)$ and assuming that $\mathcal{L}$ is Lipschitz-differentiable, the first and fifth assumptions are satisfied. As our regularization term $\mathcal{R}_c(\cdot)$ is chosen as nonconvex regularization terms such as SCAD, MCP, or ℓ_p quasi-norm, the regularization term $\mathcal{R}_c(\cdot)$ satisfy the property of restricted prox-regularity as proved in Corollary 1 in [23]. Therefore, the fourth assumption is also satisfied. Finally, because the matrix $H \cdot \Omega_2 \in \mathbb{R}^{|E| \times 2|V|}$ has full rank and $2|V| \geq |E|$, we have $\text{Im}(A_{|E|}) \subseteq \text{Im}(H \cdot \Omega_2)$, where $\text{Im}(\cdot)$ returns the image of a matrix and $A_{|E|} \in \mathbb{R}^{|E| \times |E|}$ denotes an identity matrix. Hence the second and third assumptions are also satisfied. The proof is completed. □

6 EXPERIMENTS

In this section, the performance of the proposed model FDH is evaluated using eight synthetic datasets and six real-world datasets. First, the experimental setup is introduced. The performance of the proposed model in terms of accuracy and prediction runtime is then evaluated against several existing methods. Finally, the analyses on feature computational dependency on the selected features are elaborated. All the experiments were conducted on a 64-bit machine with a quad-core processor (i7CPU@3.40GHz) and 16.0GB memory.

6.1 Experimental Settings

6.1.1 Synthetic dataset. In this experiment, 8 synthetic datasets were generated with different settings. The generation procedures were as follows. We generate the predictors of the design matrix $X \in \mathbb{R}^{20000 \times 100}$ using a Gaussian distribution with a zero mean and a standard deviation of "1". The sparse vector $W \in \mathbb{R}^{1 \times 100}$ is generated by a pairwise multiplication between a binary vector and a real-valued vector, namely $W_i = \alpha_i \cdot \beta_i$. Here each α_i is sampled from a Gaussian distribution with a mean of zero and a variance of one while β_i is sampled from a Bernoulli distribution with a probability of success of 0.5. Then the dependent variable $Y \in \{-1, 1\}^{20000 \times 1}$ is generated through the mapping $Y = \text{sign}(X \cdot W^T + \varepsilon)$, where ε is sampled from a Gaussian distribution with a mean of zero and standard a deviation of one. The basic feature cost $D \in \mathbb{R}^{200 \times 1}$ is generated from a Gaussian distribution with zero mean and a standard deviation of one. The incidence matrix $H = [H_1; H_2] \in \{0, 1\}^{200 \times 100}$ consists of two incidence matrices $H_1 \in \{0, 1\}^{100 \times 100}$ for nodes and Type-1 edges and $H_2 \in \{0, 1\}^{100 \times 100}$ for nodes and Type-2 edges. To ensure that none of the columns or rows of H_1 is an all-zero vector, $H_1 = \Phi \circ \Psi$ is generated from a pairwise "OR" operation of two binary matrices Φ and Ψ with the same size as H,

[1] http://mason.gmu.edu/~lzhao9/materials/papers/kdd2018.pdf

where $\circ$ denotes a pairwise "OR" operation, so $H_{1,i,j} = \Phi_{i,j} \bigvee \Psi_{i,j}$, Φ is an identity matrix while the elements in Ψ are randomly sampled from a Bernolli distribution with successful probability ranging from "0.1" to "0.8". This method was used to generate 8 synthetic datasets with different sparsities of the H matrix, reflecting different degrees of feature computational dependency. For all the methods, the first 5000 samples are used as the training set, the next 5000 as the validation set and the remaining 10000 as the test set.

6.1.2 Real-world datasets. Six real-world datasets for intruder detection were utilized for performance evaluation. Specifically, the network traffic-flow data of the communication signals of intruder devices in a WLAN environment were collected; 3 types of intruder devices and 2 types of network traffic mode were used for this detection. The datasets are: 1) Parrot Bebop with bidirectional traffic flow (35,143 samples); 2) DBPower UDI with bidirectional flow (31,374 samples); 3) DJI Spark with bidirectional flow (10,000 samples); 4) Parrot Bebop with unidirectional flow (21,225 samples); 5) DBPower UDI with unidirectional flow (27,024 samples); and 6) DJI Spark with unidirectional flow (132 samples). For all the datasets, the packet sizes and packet inter-arrival time are the raw data sources. For each source, the first 9 features in Table 1 are extracted. For those based on bidirectional flow, the uplink, downlink, and total traffic are considered while for those based on unidirectional flow, only the total traffic is considered. Therefore, the first three datasets have 9 features $\times$ 2 sources $\times$ 3 direction flow = 54 features and there are 9 features $\times$ 2 sources = 18 features for the remaining 3 datasets. Each sample has a label of either positive (existence of intruder) or negative (no intruder). For each dataset, both the feature generation and feature utilization runtime is measured. For the feature generation time, each feature generation runtime is computed based on the average time required to run 1000 data samples. A feature computational dependency hypergraph such as the one in Figure 1 was created and the generation time measured for each basic feature computation component (i.e., the weight of each hyperedge of Type 1) based on the average computation time for 1000 data samples. For example, for the feature "standard deviation", its feature generation runtime consists of the feature component "mean" and the remaining computation utilizing the computed "mean". The feature utilization runtime is measured as follows: we first calculate the model runtime T_a without any features (i.e., only the bias term), and then compute the model runtime T_b with all features selected. Then the feature utilization runtime (i.e., the hyperedges of Type 2) is generated as $T_b - T_a$ divided by the number of all the features. For all the datasets, the half of all the samples are randomly selected for training while the other are for testing.

6.1.3 Metrics. In this experiment, the accuracy of the classification problem was utilized as the metric for performance evaluation, namely the number of samples that are correctly classified divided by the total number of samples. Another metric is the prediction runtime, which represents the total amount of time spent on prediction including both feature generation and the model prediction using the generated features.

6.1.4 Competing Methods. This experiment utilizes 5 comparison methods: ℓ_1-regularized logistic regression [5, 29], re-weighted-ℓ_1-regularized logistic regression [1], Cost-Sensitive Tree of Classifiers (CSTC) [25], GreedyMiser [24], and Directed Acyclic Graph for cost-constrained prediction (DAG) [21]. These are described in turn below.

ℓ_1-regularized logistic regression (L1). This is a classic way to conduct cost-efficient classifications by enforcing the sparsity of the selected features. It includes a parameter that trades off the regularization term; typically, the larger this parameter is, the fewer the selected features will be.

Re-weighted-ℓ_1-regularized logistic regression (re-weighted L1). This is a generalized version of L1. Here the respective cost of each features can be considered so that features with a higher time cost will be assigned a larger penalty. Similar to L1, there is a trade-off parameter that balances empirical loss and time cost.

Cost-Sensitive Tree of Classifiers (CSTC). Similar to the re-weighted L1, this method also directly trades off the empirical loss and the runtime cost. However, this method considers the feature generation time and the feature utilization runtime separately. The trade-off parameter is tunable, as in the above two methods.

GreedyMiser. This method again trades off accuracy and feature cost in terms of the feature generation cost and the runtime of the algorithm. To approximate an optimal trade-off, an update rule based on greedy optimization is utilized with stage-wise regression.

Directed Acyclic Graph for cost-constrained prediction (DAG). This method considers the situation when several features can be budgeted together with a fixed total cost, by utilizing directed acyclic graph to search for the best combination of features.

The performance of the new methods proposed here, namely Cost-Aware classification using FCD Heterogeneous hypergraph (CAFH), was compared with the above methods. Depending on the nonconvex regularization term utilized, we have CAFH (p=1/2) and CAFH(p=2/3) where the $\ell_{1/2}$ and $\ell_{2/3}$ quasi-norms are utilized, respectively. The comparison and proposed methods were compared by extensively varying their trade-off parameters to ensure that the whole trade-off curve between runtime and accuracy is illustrated and compared.

6.2 Performance

In this section, the performance of the proposed and comparison methods are illustrated and discussed on synthetic dataset and real-world dataset in turn.

6.2.1 Performance on Synthetic Datasets. Figure 2 shows the accuracy-runtime performance curves for all the methods on the 8 synthetic datasets with increasing percentages of shared feature components. Clearly, the closer the curves are to the point (0,1), the better the performance obtained. The proposed CAFH (p=1/2) and CAFH (p=2/3) consistently outperforms the other methods on all datasets. Moreover, the advantage over other methods increases as the percentage of the shared feature components increases. This verifies the effectiveness of the proposed methods in considering and utilizing the shared feature components to reduce runtime. In contrast, the performance of the L1 and re-weighted L1 models tends to decrease as the percentage of the shared feature components increases because they are unable to take advantage of shared computations. This observation confirms that the incurred computational costs are not well reflected by those schemes, and consequently the results get steadily worse. GreedyMiser achieves a competitive performance that is better than that of L1 and re-weighted L1 but not as good as the proposed method, because it does not incorporate

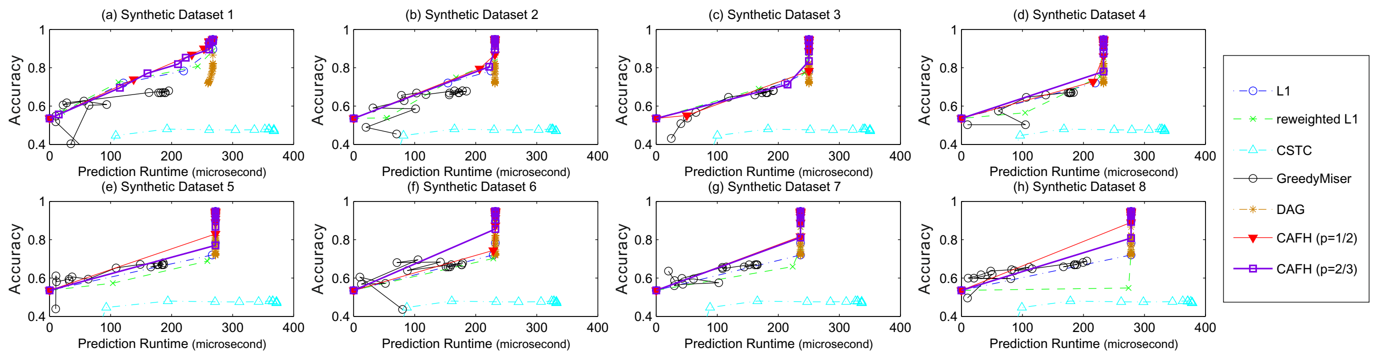

Figure 2: The Accuracy v.s. Prediction runtime per sample for all the synthetic datasets

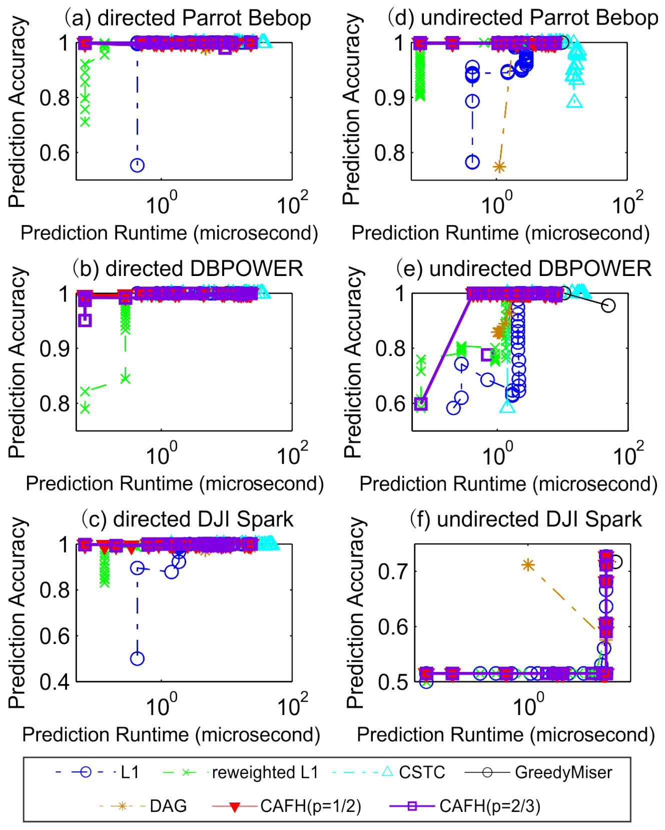

Figure 3: Accuracy v.s. prediction runtime per sample for the real world datasets

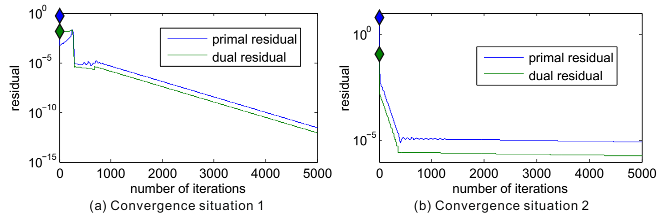

Figure 4: The convergence process in two situations, where the first one satisfies the conditions required by Theorem 5.1 while the second one does not.

feature computation dependency either. CSTC did not achieve a competitive performance on the synthetic datasets, showing that the ensemble of weak classifiers where each works with just a few features may not perform well in situations where the output is actually determined collectively by all the input features. The method DAG tended to consume more runtime when achieving the same level of accuracy as the proposed methods and some of the other competing methods because DAG does not effectively represent the costs for using individual features.

6.2.2 Performance on Real-world Datasets. Figure 3 demonstrates the effectiveness of the proposed methods CAFH (p=1/2) and CAFH (p=2/3) in all datasets, compared with the other methods. For example, for the three datasets shown in Figures 3(a), (b), and (c), the proposed CAFH obtained the best trade-off points with an accuracy of 1 and runtime of less than 0.1 millisecond, less than half of the lowest time cost with the same accuracy among the compared methods. For the datasets shown in Figure 3(d), the accuracy of the proposed CAFH is consistently 1, outperforming the accuracy obtained by the best performer among the compared methods, re-weighted L1, which drops off to 0.9 at the end. The proposed CAFH models are also among the best performers in all the methods for the last two datasets. Among all compared methods, re-weighted L1 performed best because it faithfully represents the feature utilization cost while the feature computation dependency is actually not very dense in the real-world datasets. CSTC also performed highly competitively for the same reason. DAG achieved a limited performance in our case because it is not good at considering the runtime cost for individual features. L1 failed to effectively remove costly features and wrongly dropped the key features for accurate prediction of the outputs, causing a large drop in accuracy.

6.2.3 Empirical analysis on algorithm convergence. Theorem 5.1 presents the algorithm convergence and provides the condition of convergence. By using a sufficiently large number of iterations, Figures 4(a) and 4(b) illustrate the two situations when the convergence condition is satisfied and unsatisfied, respectively. Figure 4(a) shows the linear convergence rate of the algorithm, which achieves a tiny residual of around 10^{-12} at the 5000th iteration. This contrasts markedly with the situation in Figure 4(b), where the convergence is fast with a linear rate during the first 500 iterations but then slows down and makes little further progress. However, even though the algorithm cannot achieve convergence in this situation, the residual

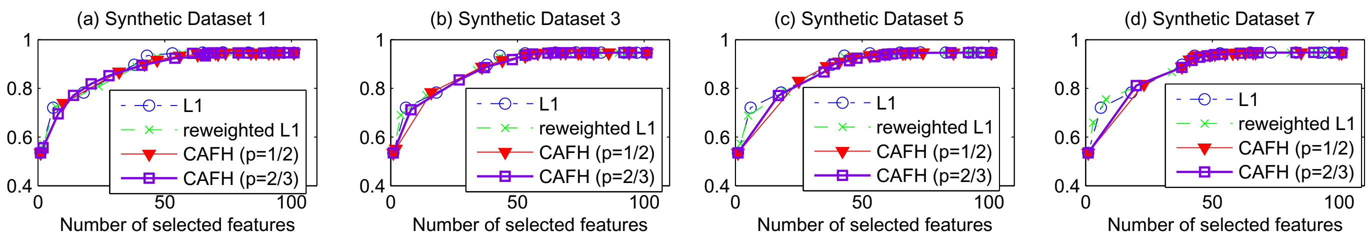

Figure 5: The Accuracy vs. number of selected features for all the synthetic datasets

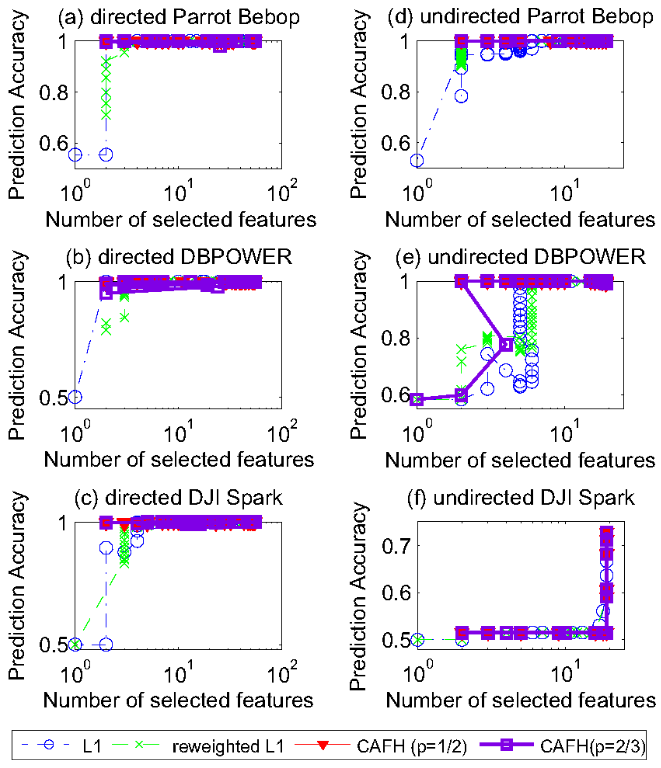

Figure 6: Accuracy vs. number of selected features for the real world datasets

Table 2: Selected features, feature computation components, and time costs (in microsecond).

Method	# selected features	# selected FCC	time cost (ms)
L1	17	82	219.6456
Reweighted L1	23	91	242.8939
CAFH (p=1/2)	22	83	209.0018
CAFH (p=2/3)	15	72	171.7118

is already less than 10^{-5}, much lower than the initial residual of around 1, which is sufficient precision for practical applications.

6.3 Analysis of Feature Computational Dependency

This section analyzes the effectiveness of the selected features in reducing runtime and retaining accuracy. First, the accuracy vs. the number of selected features is evaluated for the synthetic and real world datasets. Only 4 of all the datasets are used to show the trends for synthetic datasets due to space limitation. Next, the selected feature computational components are illustrated and analyzed.

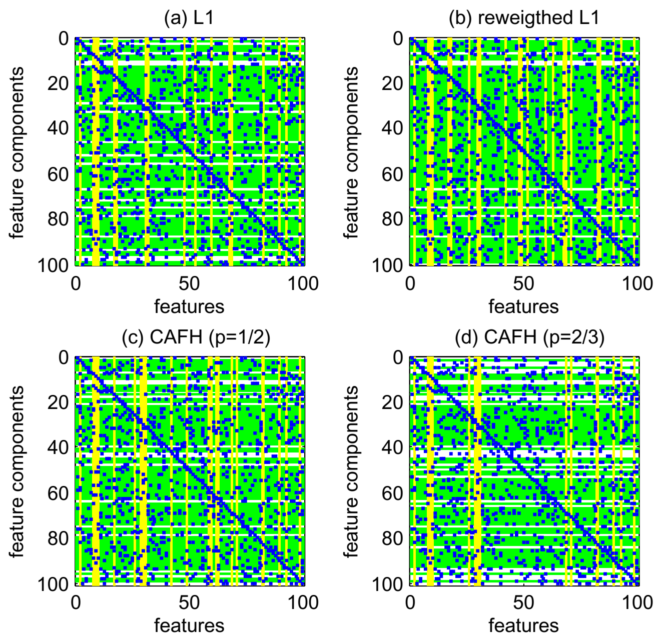

Figure 7: Selected features and the feature components they triggered

6.3.1 Accuracy vs. Number of selected features. Unlike the pattern shown in Figure 2, in Figure 5 the curves of the proposed CAFH models are typically below the curves for L1 and the re-weighted L1. This is interesting because it shows that although the proposed CAFH selects more features than L1 and the re-weighted L1, it usually incurs a smaller time cost. This confirms the effectiveness of the new approach proposed in this paper for optimizing the total time cost instead of the number of features. By selecting more features, CAFH models actually benefit from having more opportunity to include the features that are crucial for determining the outputs. When comparing Figure 6 with Figure 3, the curves for L1 tend to shift left. This shows that L1 indeed favors enforcing a small number of features, instead of the total runtime. Similarly, the re-weighted L1 also shifts left a little, especially in Figure 6(f) where it achieves a much small number of selected features. However, the total time cost was still equal to or larger than that incurred by our new model, as shown in Figure 3.

6.3.2 Selected features vs. selected feature components. Figure 7 shows the incidence matrices H between the nodes (i.e., features) and the hyperedges (i.e., feature computation components). The incidence matrices are the same for all four subplots, where blue points denote the nonzero elements in the matrix. When a feature is selected, the corresponding column is shown in yellow, and the rows (i.e., feature computation components) where the blue points interact with the yellow columns are marked in green. This is actually just the selected feature computation components based on the selected features. Table 2 shows the number of selected features and the number of selected FCCs in Figure 7. Both Table 2 and Figure 7 show that when selecting almost the same number of features, the proposed CAFH methods tend to select fewer FCC, showing that the CAFH models do better at exploiting shared computations among the features to reduce the runtime cost.

7 CONCLUSIONS

To effectively reduce the prediction-time cost and retain the model accuracy, this paper proposes a novel framework that jointly minimizes the classification error and the prediction runtime cost by considering feature computational dependency. We first model feature computational dependency as FCD heterogeneous hypergraph and propose a concise objective function with a faithful representation of the runtime costs. To optimize this objective function which consists of both continuous and discrete parts, we propose a tight continuous approximation to the original problem and an ADMM-based algorithm to solve it effectively with a convergence guarantee under specific conditions. Extensive experiments on several synthetic and real-world datasets demonstrated the advantageous performance of the proposed model over the existing methods for cost-sensitive classification. Detailed analysis on the selected features and FCCs were also presented to show the effectiveness of the proposed method.

ACKNOWLEGEMENT

This work was supported by the National Science Foundation grant: #1755850.

REFERENCES

[1] Aleksandr Y Aravkin, James V Burke, and Gianluigi Pillonetto. 2013. Sparse/robust estimation and kalman smoothing with nonsmooth log-concave densities: Modeling, computation, and theory. *The Journal of Machine Learning Research* 14, 1 (2013), 2689–2728.

[2] Rick Chartrand and Wotao Yin. 2016. Nonconvex sparse regularization and splitting algorithms. In *Splitting Methods in Communication, Imaging, Science, and Engineering*. Springer, 237–249.

[3] Lingwei Chen, Shifu Hou, and Yanfang Ye. 2017. SecureDroid: Enhancing Security of Machine Learning-based Detection Against Adversarial Android Malware Attacks. In *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC 2017)*. ACM, New York, NY, USA, 362–372. https://doi.org/10.1145/3134600.3134636

[4] Kun Deng, Yaling Zheng, Chris Bourke, Stephen Scott, and Julie Masciale. 2013. New algorithms for budgeted learning. *Machine learning* 90, 1 (2013), 59–90.

[5] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. 2004. Least angle regression. *The Annals of statistics* 32, 2 (2004), 407–499.

[6] Yujie Fan, Yanfang Ye, and Lifei Chen. 2016. Malicious sequential pattern mining for automatic malware detection. *Expert Systems with Applications* 52 (2016), 16 – 25. https://doi.org/10.1016/j.eswa.2016.01.002

[7] César Ferri, Peter Flach, and José Hernández-Orallo. 2002. Learning decision trees using the area under the ROC curve. In *ICML*, Vol. 2. 139–146.

[8] Alex Grubb and Drew Bagnell. 2012. Speedboost: Anytime prediction with uniform near-optimality. In *Artificial Intelligence and Statistics*. 458–466.

[9] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. 2017. HinDroid: An Intelligent Android Malware Detection System Based on Structured Heterogeneous Information Network. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM, New York, NY, USA, 1507–1515. https://doi.org/10.1145/3097983.3098026

[10] Cheng-Lung Huang and Chieh-Jen Wang. 2006. A GA-based feature selection and parameters optimizationfor support vector machines. *Expert Systems with applications* 31, 2 (2006), 231–240.

[11] Aloak Kapoor and Russell Greiner. 2005. Learning and classifying under hard budgets. In *European Conference on Machine Learning*. Springer, 170–181.

[12] Matt J Kusner, Wenlin Chen, Quan Zhou, Zhixiang Eddie Xu, Kilian Q Weinberger, and Yixin Chen. 2014. Feature-Cost Sensitive Learning with Submodular Trees of Classifiers.. In *AAAI*. 1939–1945.

[13] Liyun Li, Umut Topkara, Baris Coskun, and Nasir Memon. 2009. CoCoST: a computational cost efficient classifier. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*. IEEE, 268–277.

[14] Liyun Li, Umut Topkara, and Nasir Memon. 2011. ACE-Cost: acquisition cost efficient classifier by hybrid decision tree with local SVM leaves. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer, 60–74.

[15] Charles X Ling, Victor S Sheng, and Qiang Yang. 2006. Test strategies for cost-sensitive decision trees. *IEEE Transactions on Knowledge and Data Engineering* 18, 8 (2006), 1055–1067.

[16] Feng Nan and Venkatesh Saligrama. 2017. Adaptive Classification for Prediction Under a Budget. In *Advances in Neural Information Processing Systems*. 4730–4740.

[17] Junbiao Pang, Huihuang Lin, Li Su, Chunjie Zhang, Weigang Zhang, Lijuan Duan, Qingming Huang, and Baocai Yin. 2016. Accelerate convolutional neural networks for binary classification via cascading cost-sensitive feature. In *Image Processing (ICIP), 2016 IEEE International Conference on*. IEEE, 1037–1041.

[18] Jay Pujara, Hal Daumé III, and Lise Getoor. 2011. Using classifier cascades for scalable e-mail classification. In *Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference*. ACM, 55–63.

[19] Lev Reyzin. 2011. Boosting on a budget: Sampling for feature-efficient prediction. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. Citeseer, 529–536.

[20] Ming Tan and Jeffrey C Schlimmer. 1989. Cost-sensitive concept learning of sensor use in approach and recognition. In *Proceedings of the sixth international workshop on Machine learning*. Elsevier, 392–395.

[21] Joseph Wang, Kirill Trapeznikov, and Venkatesh Saligrama. 2015. Efficient learning by directed acyclic graph for resource constrained prediction. In *Advances in Neural Information Processing Systems*. 2152–2160.

[22] Junxiang Wang and Liang Zhao. 2018. Multi-instance Domain Adaptation for Vaccine Adverse Event Detection. In *Proceedings of the 2018 World Wide Web Conference (WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 97–106. https://doi.org/10.1145/3178876.3186051

[23] Yu Wang, Wotao Yin, and Jinshan Zeng. 2015. Global convergence of ADMM in nonconvex nonsmooth optimization. *arXiv preprint arXiv:1511.06324* (2015).

[24] Zhixiang Xu, Kilian Q Weinberger, and Olivier Chapelle. 2012. The greedy miser: learning under test-time budgets. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*. Omnipress, 1299–1306.

[25] Zhixiang Eddie Xu, Matt J Kusner, Kilian Q Weinberger, Minmin Chen, and Olivier Chapelle. 2014. Classifier cascades and trees for minimizing feature evaluation cost. *Journal of Machine Learning Research* 15, 1 (2014), 2113–2144.

[26] Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. 2017. A Survey on Malware Detection Using Data Mining Techniques. *ACM Comput. Surv.* 50, 3, Article 41 (June 2017), 40 pages. https://doi.org/10.1145/3073559

[27] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*. USENIX Association, Berkeley, CA, USA, 2–2. http://dl.acm.org/citation.cfm?id=2228298.2228301

[28] Liang Zhao, Qian Sun, Jieping Ye, Feng Chen, Chang-Tien Lu, and Naren Ramakrishnan. 2015. Multi-task learning for spatio-temporal event forecasting. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1503–1512.

[29] Liang Zhao, Qian Sun, Jieping Ye, Feng Chen, Chang-Tien Lu, and Naren Ramakrishnan. 2015. Multi-Task Learning for Spatio-Temporal Event Forecasting. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*. ACM, New York, NY, USA, 1503–1512. https://doi.org/10.1145/2783258.2783377

[30] Liang Zhao, Jieping Ye, Feng Chen, Chang-Tien Lu, and Naren Ramakrishnan. 2016. Hierarchical Incomplete Multi-source Feature Learning for Spatiotemporal Event Forecasting. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 2085–2094. https://doi.org/10.1145/2939672.2939847

[31] W. Zhuang, Y. Ye, Y. Chen, and T. Li. 2012. Ensemble Clustering for Internet Security Applications. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 6 (Nov 2012), 1784–1796. https://doi.org/10.1109/TSMCC.2012.2222025