Large-scale Cost-aware Classification Using Feature Computational Dependency Graph

Qingzhe Li, Amir Alipour-Fanid, Martin Slawski,Yanfang Ye, Lingfei Wu, Kai Zeng, Liang Zhao

Abstract—With the rapid growth of real-time machine learning applications, the process of feature selection and model optimization requires to integrate with the constraints on computational budgets. A specific computational resource in this regard is the time needed for evaluating predictions on test instances. The joint optimization problem of prediction accuracy and prediction-time efficiency draws more and more attention in the data mining and machine learning communities. The runtime cost is dominated by the feature generation process that contains significantly redundant computations across different features that sharing the same computational component in practice. Eliminating such redundancies would obviously reduce the time costs in the feature generation process. Our previous Cost-aware classification using Feature computational dependencies heterogeneous Hypergraph (CAFH) model has achieved excellent performance on the effectiveness. In the big data era, the high dimensionality caused by the heterogeneous data sources leads to the difficulty in fitting the entire hypergraph into the main memory and the high computational cost during the optimization process. Simply partitioning the features into batches cannot give the optimal solution since it will lose some feature dependencies across the batches. To improve the high memory and computational costs in the CAFH model, we propose an equivalent Accelerated CAFH (ACAFH) model based on the lossless heterogeneous hypergraph decomposition. An efficient and effective nonconvex optimization algorithm based on the alternating direction method of multipliers (ADMM) is developed to optimize the ACAFH model. The time and space complexities of the optimization algorithm for the ACAFH model are three and one polynomial degrees less than our previous algorithm for the CAFH model, respectively. Extensive experiments demonstrate the proposed ACAFH model achieves competitive performance on the effectiveness and much better performance on the efficiency.

Index Terms—Feature Computational Dependency, Cost-sensitive Learning

1 Introduction

With the rapid growth of heterogeneous data sources, an explosive number of new features become available. Handling high-dimensional data is one of the major challenges of the machine learning methods in many real-world applications, including earthquake detection [1], [2], adult content filtering [3], and intruder detection [4]. Additional requirements from industrial fields must be taken into account, especially the timeliness of the prediction [5]. Moreover, given that Google processes over 40,000 search queries every second on average in 2018, running a machine learning algorithm on a high-dimensional dataset 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 [6]. 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 cost of feature extraction dominates the test-time runtime cost, especially when linear models such as those commonly used in industrial settings are utilized [7], [8], [9], [10], [11]. 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 [12], [13], [14]. Explicit methods optimize criteria involving a trade-off between prediction accuracy and prediction cost [15], [16], [17]. 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.

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]	$MAD = median(\|x(i) - median(x)\|)$	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	$H = (Max(x(i))\|_{i=1...N})$	0.464 microsecond
V_8: MIN	$L = (Min(x(i))\|_{i=1...N})$	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

In a number of important applications, some computations for generating different features tend to be shared [18], whereupon we refer to their generation processes as being ***computationally 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

• *Corresponding Author: Qingzhe Li (qli10@gmu.edu) & Liang Zhao (lzhao9@gmu.edu)*

1. MAD: median absolute deviation; STD: standard deviation

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. **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. **4) Inefficiency of optimizing the model with tremendous and highly sparse feature computation dependencies.** With the rapid growth of heterogeneous data sources, the dimensionality of the data could be very large, which causes the difficulty of fitting all feature dependencies into the main memory during the optimization process. Simply partitioning the features into batches cannot give the optimal solution since it will lose some feature dependencies across the batches. Therefore, an effective and scalable partition way is desired without losing the feature dependencies during the partitioning process.

Although our previously proposed Cost-Aware classification using Feature computational dependency heterogeneous Hypergraph (CAFH) model [19] outperformed other methods on the effectiveness, its efficiency on optimizing the model still needs to be improved. Specifically, the space and time complexity of optimizing the CAFH model are respectively $O(m^2)$ and $O(m^4)$, where m is the number of the features. To address the inefficiency of optimizing the CAFH model, in this paper, we extend the CAFH model by proposing the Accelerated CAFH (ACAFH) model, whose effectiveness is similar to the CAFH model but can be trained much faster with much less memory usage. The major contributions of this paper are summarized as follows:

- We propose the ACAFH model based on heterogeneous hypergraph decomposition and our previously proposed CAFH model. The ACAFH improves the efficiency without compromising the effectiveness while handling the cost-aware classification problem under the high dimensionality setting caused by multiple heterogeneous data sources.
- We theoretically analyze the improvements in terms of the space and time complexities while optimizing the proposed ACAFH model. In addition, we prove the equivalence between the CAFH and ACAFH models on their optimization objectives.
- We develop an effective algorithm to solve the non-convex problems in the proposed ACAFH model with theoretical guarantees. The proposed algorithm is effective and scalable to high dimensional datasets.
- We conduct extensive experiments on eight synthetic datasets and six real-world datasets to show the improvements on the efficiency of the proposed algorithm. In addition, we also validate the similar state-of-the-art performance on effectiveness between the CAFH and ACAFH models.

2 Related Work

In this section, we first introduce our previously proposed Cost-Aware classification using FCD heterogeneous Hypergraph (CAFH) methods. Then we briefly review test-time cost-efficient models, which can be categorized into implicit methods, explicit methods, budgeted learning models, cost-aware data acquisition, and sparse feature learning models.

Our previously proposed CAFH method: In our previous work [19], we solved the cost-efficient machine learning problem as an optimization problem, which minimized the prediction error and runtime cost by optimally utilizing the Feature Computational Dependencies (FCD) among the different features. Specifically, we proposed to represent the FCDs as a heterogeneous hypergraph. Then we proposed the CAFH model that embedded the hypergraph representation of FCD. We also proposed a tight relaxation of the original problem to solve the discontinuous optimization problem in the CAFH model. Finally, we developed a non-convex optimization algorithm based on the Alternating Direction Methods of Multipliers (ADMM) [20], [21], [22], whose effectiveness was validated on both synthetic and real-world datasets. However, when the number of features is large and come from different data sources, the CAFH method is the inefficiency in terms of the space and time complexity during the training phase.

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 [12], [23], [24]. 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 [25]. For example, Tan and Schlimmer [14] employed an entropy-based strategy to estimate the cost while Ferri et al. [26] leveraged the feature cost to prune the tree after it had been built. Li et al. [7] 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. [13] 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 [27]. To consider the different costs of the various features, a number of approaches have been proposed. For example, Grubb et al. [15] proposed an algorithm for "anytime prediction" which outputs predictions with increasing quality as the cost budget increases. Xu et al. [28] developed Greedy Miser, a variant of regular stage-wise regression, which updates the selected features using a greedy optimization strategy. Kusner et al. [17] formulated the cost-sensitive feature selection as an approximate submodular optimization problem, while Huang and Wang [16] 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 [29]. For example, Deng et al. [30] designed algorithms for the multi-armed bandit problem to select specific features for specific instances under a limited budget. Nan and Saligrama [31] 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. [32] 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 and Our Previous Model

In this section, we first introduce the notations and present the cost-aware classification problem, then we briefly review how our previously model solve cost-aware classification problem.

3.1 Problem Setup

Define $X = \{X_1, X_2, \cdots, X_n\} \in \mathbb{R}^{n\times m}$ as the input data containing n samples under m features, where each sample is a row vector $X_j \in \mathbb{R}^{1\times m}$. Denote $X_{j,i} \in \mathbb{R}$ as the element at index i of X_j which corresponds to a feature $v_i \in V$, where $V = \{v_1, \cdots, v_m\}$. In addition, denote $X_{j,\boldsymbol{\alpha_1}} = \{X_{j,i} | i \in \boldsymbol{\alpha_1}\}$ as the subspace vector of X_j where the Greek letter in the subscript, i.e., $\boldsymbol{\alpha_1}$, denotes a subset of indices. Similarly, $V_{\boldsymbol{\alpha_1}} = \{v_i | i \in \boldsymbol{\alpha_1}\}$ denotes the subset of the features define by the subset of the indices $\boldsymbol{\alpha_1}$. 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:

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

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 The Accelerated CAFH Model

To solve the cost-aware classification problem with high dimensionality and sparse feature computational dependencies, we propose the Accelerated Cost-Aware classification using large-scale sparse FCD heterogeneous Hypergraph (ACAFH) model in this section. The ACAFH model is equivalent to our CAFH model but can be optimized much more efficiently using less time and memory spaces. Specifically, we first decompose the large-scale sparse FCD heterogeneous hypergraph into several Connected Components in Heterogeneous Hypergraph (CCHH). Then we propose our ACAFH model by using the CCHHs instead of the entire heterogeneous hypergraph. After that, the original discrete and non-convex problem in ACAFH is transformed to its continuous equivalence. Finally, a more efficient ADMM-based algorithm is proposed and its time and space complexity are analyzed.

4.1 Large-scale Sparse FCD Heterogeneous Hypergraph Decomposition

In this section, we first recall the heterogeneous hypergraph modeling of the feature computational dependency to make this paper complete. Then, the subproblem of large-scale sparse FCD heterogeneous hypergraph decomposition is formulated. Finally, an efficient algorithm is presented to solve the subproblem.

4.1.1 The heterogeneous hypergraph modeling for FCD

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 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:

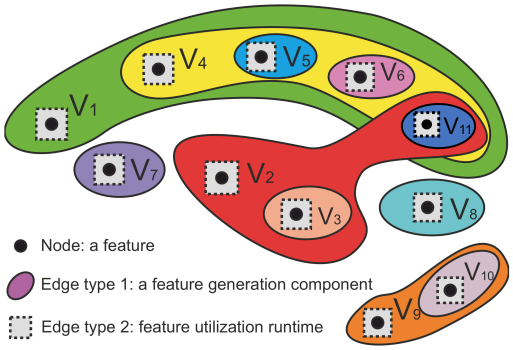


Fig. 1: An overview of the feature computational dependency heterogeneous hypergraph for Table 1.

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. Several nodes can be linked by the same Type-1 hyperedge if they share the same FCC; 2) Type 2: FCCs for feature utilization. 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. $w(E) \in \mathbb{R}^{d\times 1}$ denotes the weights of all the hyperedges and d is the cardinality of E. The weight of hyperedge e_i is represented as $w(e_i)$, denoting time cost for the corresponding FCC. In addition, all the FCDs is denoted by an incidence matrix $H \in \{0, 1\}^{d\times m}$ between the nodes and hyperedges, where $H_{i,j} = 1$ 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.*

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 11 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 7 *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.

To reduce the optimization cost in CAFH for large-scale sparse hypergraph, we propose to decompose the FCD Het-

Algorithm 1 FCD Heterogeneous Hypergraph Decomposition

Require: The incidence matrix H.
Ensure: 1. the number of CCHHs K;
2. the partition of the hyperedges' indices: $\{\boldsymbol{\alpha}_k\}_{k=1}^K$;
3. the partition of the features' indices $\{\boldsymbol{\beta}_k\}_{k=1}^K$

1: global: $(d, m) \leftarrow$ size(H)
2: global: $exploredFeatures \leftarrow 0^{m\times 1}$
3: global: $mapFCC2CC \leftarrow 0^{d\times 1}$
4: $k \leftarrow 0$ // the count of current CCHHs
5: global: $adjcentListFCC, adjecentListFeature \leftarrow$ toAdjcentList(H)
6: **for** $row \leftarrow 1, \cdots, d$ **do**
7: **if** $mapFCC2CC == 0$ **then**
8: $k \leftarrow k + 1$
9: global: $\boldsymbol{\alpha}_k \leftarrow \emptyset$
10: global: $\boldsymbol{\beta}_k \leftarrow \emptyset$
11: DFS(row, k)
12: **end if**
13: **end for**
14: $K \leftarrow k$
15: **return** $K, \boldsymbol{\alpha}, \boldsymbol{\beta}$
16: **procedure:** DFS(row, k)
17: **if** $mapFCC2CC(row) == 0$ **then**
18: $mapFCC2CC(row) \leftarrow k$
19: $\boldsymbol{\alpha}_k$.add(row)
20: **for** $feature$ in $adjcentListFCC(row)$ **do**
21: **if** $exploredFeatures(feature) == 0$ **then**
22: $\boldsymbol{\beta}_k$.add$(feature)$
23: $exploredFeatures(feature) \leftarrow 1$
24: **for** fcc in $adjcentListFeature(feature)$ **do**
25: **if** $mapFCC2CC(fcc) == 0$ **then**
26: DFS(fcc, k)
27: **end if**
28: **end for**
29: **end if**
30: **end for**
31: **end if**
32: **end procedure**

erogeneous Hypergraph $\mathcal{G}(V, E, w(E))$ into several maximal non-overlapping connected sub-hypergraphs, each is called a *connected component in heterogeneous hypergraph*. Before we formally formulate the FCD heterogeneous hypergraph decomposition problem, a few concepts need to be defined.

Definition 2 (hyperpath)**.** *Given a heterogeneous hypergraph $\mathcal{G} = (V, E, w(E))$. A **hyperpath** exists between two vertices $\{v_1, v_2\} \in V$ if and only if either v_1 and v_2 share a hyperedge or exists v_i such that v_1 and v_i share a hyperedge, there exists a hyperpath between v_i and v_2.*

Definition 3 (Connected Component in Heterogeneous Hypergraph (CCHH))**.** *A **connected component in heterogeneous hypergraph** $\mathcal{G}_k = (V_{\boldsymbol{\beta}_k}, E_{\boldsymbol{\alpha}_k}, w(E_{\boldsymbol{\alpha}_k}))$, where $V_{\boldsymbol{\beta}_k} \subseteq V$, and $|V_{\boldsymbol{\beta}_k}| = m_k$, $E_{\boldsymbol{\alpha}_k} \subseteq E$ and $|E_{\boldsymbol{\alpha}_k}| = d_k$, is defined as a maximal connected sub-graph of $\mathcal{G}$, such that: 1). for all $\{v_i, v_j\} \in V_{\boldsymbol{\beta}_k}$ there exists a hyperpath between v_i and v_j, and 2). for any $v_i \in V_{\boldsymbol{\beta}_k}$ and for any $v_q \notin V_{\boldsymbol{\beta}_k}$ there does not exist any hyperpaths between v_i and v_q. The subset of the features $V_{\boldsymbol{\beta}_k}$ is determined by subset of the features' indices $\boldsymbol{\beta}_k \subseteq \{1, 2, \cdots, m\}$, and the subset of the hyperedges $E_{\boldsymbol{\alpha}_k}$ and their weights $w(E_{\boldsymbol{\alpha}_k})$ are determined by the subset of hyperedges' indices $\boldsymbol{\alpha}_k \subseteq \{1, 2, \cdots, d\}$.*

To find all the CCHHs in $\mathcal{G}$, the problem of *FCD Heterogeneous Hypergraph Decomposition* is formulated as follows:
Problem Formulation: Given a FCD heterogeneous hypergraph $\mathcal{G}$ and its incidence matrix $H \in \{0, 1\}^{d\times m}$, the goal is to find a partition way characterized by a finite number of *disjoint* subsets $\{\boldsymbol{\alpha}_k\}_{k=1}^K$ and $\{\boldsymbol{\beta}_k\}_{k=1}^K$ such that $\bigcup_{k=1}^K \boldsymbol{\alpha}_k = \{1, 2, \cdots, d\}$, $\bigcup_{k=1}^K \boldsymbol{\beta}_k = \{1, 2, \cdots, m\}$, and every resulting subhypergraph $\mathcal{G}_k = (V_{\boldsymbol{\beta}_k}, E_{\boldsymbol{\alpha}_k}, w(E_{\boldsymbol{\alpha}_k})$ is a CCHH whose incidence matrix is $H_{\boldsymbol{\alpha}_k, \boldsymbol{\beta}_k} \in \{0, 1\}^{d_k\times m_k}$, where K is the number of all CCHHs to be determined.

The optimal solution of the above problem can be found by Algorithm 1 in $O(md)$ time based on a Deep-First-Search (DFS) strategy. Specifically, after initializing the global variables in Lines 1-4, Line 5 converts the incidence matrix H to two adjacent lists in $O(md)$ time. To find the all the CCHHs in $\mathcal{G}$, Lines 6-13 traverse all FCCs and features that have not yet been explored by using a DFS algorithm which is elaborated in Lines 16-32. By taking the advantages of the adjacent list data structure, the entire traversal takes only $O(md)$ time. Therefore, the time complexity of Algorithm 1 is still $O(md)$.

4.2 Accelerated Cost-aware classification using FCD Heterogeneous Hypergraph (ACAFH)

In this section, we propose the Accelerated Cost-aware classification using FCD Heterogeneous Hypergraph (ACAFH) model that is euqivelent to our CAFH model but can be optimized more efficiently.

To simplify the notations, we directly use V_k, E_k, and H_k to denote $V_{\boldsymbol{\beta}_k}$, $E_{\boldsymbol{\alpha}_k}$ and $H_{\boldsymbol{\alpha}_k, \boldsymbol{\beta}_k}$ in the rest of this paper. Following this notation style, we denote the feature weights as $W_k = W_{\boldsymbol{\beta}_k} \in \mathbb{R}^{m_k\times 1}$, the time costs as $D_k = D_{\boldsymbol{\alpha}_k} \in \mathbb{R}^{d_k\times 1}$.

The selection of a feature is indicated by its weight: $W_{k,i} = 0$ means feature $v_i \in V_k$ is not being used and thus can be ignored in the prediction phase, and $W_{k,i} \neq 0$ means it is included. Similarly, the selected features $U_k \subseteq V_k$ is defined as $U_k = \{v_i | I(W_{k,i}) = 1, v_i \in V_k\}$, where the indicator function $I(W_{k,i}) = 0$ when $W_{k,i} = 0$ and $I(W_{k,i}) = 1$ when $W_{k,i} \neq 0$.

The FCCs associated with the features in U_k are ${E_k}' = \bigcup_v^{U_k} e(v) \subseteq E_k$. As $\{\boldsymbol{\alpha_k}\}_{k=1}^K$ is a partition way of the index set of the hyperedges, for any $k' \neq k$, we have $E_k \cap E_{k'} = \emptyset$ and $E = \bigcup_{k=1}^K E_k$. In addition, since ${E_k}' \subseteq E_k$, we also have ${E_i}' \cap {E_j}' = \emptyset$ and the selected hyperedges (i.e. FCCs) in the hypergraph are equal to the union of the selected hyperedges in each CCHH: $E' = \bigcup_{k=1}^K {E_k}'$. Therefore, the total runtime is: $\mathcal{T}_1(U; \mathcal{G}) + \mathcal{T}_2(U) = \sum_e^{E'} w(e) = \sum_{k=1}^K \sum_e^{{E_k}'} w(e)$. By using the matrix representation, the total runtime can be rewritten as:

$$\mathcal{T}_1(U; \mathcal{G}) + \mathcal{T}_2(U) = \sum_{k=1}^K \sum_e^{{E_k}'} w(e) = \sum_{k=1}^K (D_k)^T \cdot \mathbf{I}(H_k \cdot \mathbf{I}(W_k)),$$

where the $\mathbf{I}(\cdot)$ is the indicator function for vectors, which maps all non-zero elements to 1 and maps all zero elements to 0. $D \in \mathbb{R}^{d\times 1}$ is a vector whose elements are the weights of the hyperedges in FCD heterogeneous hypergraph, namely $D_{k,i} = w(e_i)$ for each ith hyperedge $e_i \in E_k$. In practice, typically $\mathcal{T}_1 \gg \mathcal{T}_2$ [33], so that in some practical applications we may only need to consider Type 1 edges. 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:

$$\min_W \mathcal{L}(W), s.t. \sum_k^K (D_k)^T \cdot \mathbf{I}(H_k \cdot \mathbf{I}(W_k)) \leq \tau - const, \quad (3)$$

$$\min_W \mathcal{L}(W) + \lambda \cdot \sum_k^K (D_k)^T \cdot \mathbf{I}(H_k \cdot \mathbf{I}(W_k)), \quad (4)$$

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 ACAFH Optimization Objective

To optimize Equation (3) and (4), which are discrete and nonconvex, we propose to transform the original objective function for ACAFH to a relaxed objective function with a set of convex constraints. As the formulations in Equations (3) and (4) are equivalent, we will focus on the regularization form (4) in the following.

First, Equation (4) can be simplified equivalently by replacing the inside indicator function with an element-wise absolute-value operation:

$$\min_W \mathcal{L}(W) + \lambda \cdot \sum_k^K D_k^T \cdot \mathbf{I}(H_k \cdot |W_k|) \tag{5}$$

where $|W_k| \in \mathbb{R}^{m_k \times 1}$ is an absolute valued vector of W_k. Then, $\mathbf{I}(H_k \cdot |W_k|)$ in Equation (5) is replaced by a simple indicator function in an auxiliary variable $M_k \in \mathbb{R}^{d_k \times 1}$:

$$\min_{W,M} \mathcal{L}(W) + \lambda \cdot \sum_k^K (D_k)^T \cdot \mathbf{I}(M_k),\ s.t.,\ M_k = H_k \cdot |W_k| \tag{6}$$

which has two nonconvex parts due to the indicator function as well as the nonlinear equality constraint. To solve this problem, an effective continuous relaxation with convex-equivalent constraint is derived below.

4.3.1 Continuous relaxation of ACAFH's 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 (5) or (6) into the absolute value function, one obtains the optimization problem

$$\min_W \mathcal{L}(W) + \lambda \cdot \sum_k^K D_k^T \cdot |H_k \cdot |W_k||$$

which entails that the FCCs associated with each row of H_k are weighted by the ℓ_1-norm of W_k; 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.

$$\begin{aligned} \min_{W,\{M_k\}_{k=1}^K} \mathcal{L}(W) + \lambda \cdot \sum_{k=1}^K \mathcal{R}_c(M_k, D_k) \\ \text{s.t. }, M_k = H_k \cdot |W_k|, \forall k = 1 \dots K \end{aligned} \tag{7}$$

where $\mathcal{R}_c(M_k, D_k)$ denotes a re-weighted version of the nonconvex regularization term such that $\mathcal{R}_c(M_k, D_k) = \sum_i^{d_k} D_{k,i} \cdot \mathcal{R}(M_{k,i})$. Here $\mathcal{R}(\cdot)$ can be a commonly used concave regularization term such as MCP, SCAD, and ℓ_p quasi-norms $(0 < p < 1)$ [34]. For example, when we use re-weighted ℓ_p quasi-norms, then $\mathcal{R}_c(M_k, D_k) = \|\text{diag}(D_k^{1/p}) \cdot M_k\|_p^p$, which is easy to compute and also satisfies the triangle inequality. Here $\text{diag}(D_k^{1/p})$ denotes the diagonal matrix whose diagonal elements are the vector pth root of D_k, namely $[\text{diag}(D_k^{1/p})]_{i,i} = D_{k,i}^{1/p}$. 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

Equation (7) contains K nonconvex constraints due to the non-linearity of $|W_k|$. For the kth CCHH, by introducing the auxiliary variables $B_k^+ \in \mathbb{R}^{m_k \times 1}$ and $B_k^- \in \mathbb{R}^{m_k \times 1}$ whose ith elements are defined as $B_{k,i}^+ = \max(0, W_{k,i})$ and $B_{k,i}^- = \max(-W_{k,i}, 0)$, respectively. Therefore, $W_k = B_k^+ - B_k^-$, and we have $|W_k| = B_k^+ + B_k^-$ if given $B_{k,i}^+ \cdot B_{k,i}^- = 0$, for any $i = 1, \cdots, m_k$. Using matrix notation, we can denote $|W_k| = \hat{\Omega}_k \cdot B_k$, where $B_k = [B_k{}^+; B_k{}^-] \in \mathbb{R}^{2m_k \times 1}$, $\hat{\Omega}_k = [A_k, A_k] \in \{0,1\}^{m_k \times 2m_k}$ where A_k denotes the identity matrix of size $m_k \times m_k$. Similarly, we can replace W by $\Omega \cdot B$ where $B = [B^+; B^-] \in \mathbb{R}^{2m \times 1}$ and $\Omega \in [A, -A] \in \{0,1\}^{m \times 2m}$.

Therefore, Equation (7) is transformed into the following, which is also the optimization objctive function of our ACAFH model:

$$\begin{aligned} \min_{M,B \geq 0} \mathcal{L}(\Omega \cdot B) + \lambda \sum_k^K \cdot \mathcal{R}_c(M_k, D_k) \\ s.t., M_k = H_k \cdot \hat{\Omega}_k \cdot B_k, \forall k = 1 \dots K \end{aligned} \tag{8}$$

This indicates that the feasible set constrained by the equality constraint in Equation (8) is convex. Now we can formally state and prove the equivalence of Equations (7) and (8):

Theorem 1 (Equivalence of Formulations (8) and (7))**.** *When $\mathcal{R}_c$ is strictly monotonically increasing in $[0, +\infty)$, formulations* (8) *and* (7) *are equivalent. When $\mathcal{R}_c$ is monotonically non-decreasing in $[0, +\infty)$, the optimal solution of Equation* (8) *must be the optimal solution of Equation* (7)*.*

Proof. The proof amounts to proving there does not exist any integer pair (k', i), where $1 \leq k' \leq K$, $1 \leq i \leq m_{k'}$, and $B_{k',i}^+ \cdot B_{k',i}^- \neq 0$, such that $B = \{[B_{k'}^+; B_{k'}^-]\}_{k'}^K$ in Equation (8) is also the optimal solution in Equation (7). We prove by contradiction below. Assume there exists one and only one pair of integers $\{k', i | 1 \leq k' \leq K \text{ AND } 1 \leq i \leq m_{k'} \text{ AND } B_{k',i}^+ > 0 \text{ AND } B_{k',i}^- > 0 \text{ AND } B_{k',i}^+ \cdot B_{k',i}^- \neq 0\}$, such that $B = \{[B_{k'}^+; B_{k'}^-]\}_{k'}^K$ is the optimal solution for both Equation (7) and Equation (8). Now we show the contradiction by first constructing a solution $B' = \{[B'^+_{k'}; B'^-_{k'}]\}_{k'}^K$ and then prove B' is better (when $\mathcal{R}_c$ is strictly monotonically increasing) or not worse (when $\mathcal{R}_c$ is monotonically non-decreasing) than B.

We construct B' as follows: we first let

$$[B'^+_{k',i}; B'^-_{k',i}] = \begin{cases} [B_{k,i}^+ - B_{k,i}^-; 0] & \text{, if } B_{k',i}^+ \geq B_{k',i}^- \\ [0; B_{k,i}^- - B_{k,i}^+] & \text{, if } B_{k',i}^+ < B_{k',i}^- \end{cases}$$

Then for any other integer pair $\{b, j | (b \neq k' \text{ OR } j \neq i) \text{ AND } 1 \leq b \leq K, \text{ AND } 1 \leq j \leq m_b\}$, we simply let $[B'^+_{b,j}; B'^-_{b,j}] = [B_{b,j}^+; B_{b,j}^-]$.

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 $\Omega \cdot B \equiv \Omega \cdot B'$. Base on our assumption of $B_{k',i}^+ > 0$, $B_{k',i}^- > 0$, and $B_{k',i}^+ \cdot B_{k',i}^- \neq 0$, it is always true for $B'^+_{k',i} + B'^-_{k',i} < B_{k',i}^+ + B_{k',i}^-$. When $j \neq i$, we have $B'^+_{k',j} + B'^-_{k',j} \equiv B_{k',j}^+ + B_{k',j}^-$. In addition, since $H_{k',i} \geq 0$ and none of its column is all-zeros, which indicates there exists at least one integer $r = 1, \cdots, d_{k'}$ such that satisfies $H_{k',r,i} = 1$. Therefore, we have $M'_{k',r} = \sum_{a=1}^{m_{k'}} H_{k',r,a} \cdot \hat{\Omega}_{k',a,\cdot} \cdot B'_{k'} < \sum_{a=1}^{m_{k'}} H_{k',r,a} \cdot \hat{\Omega}_{k',a,\cdot} \cdot B_{k'} = M_{k'}$, where $\hat{\Omega}_{k',a,\cdot}$ denotes the ath row of $\hat{\Omega}_{k'}$. For any $b \neq k'$, by definition, we have $M'_b \equiv M_b$.

Finally, we have $\sum_k^K \mathcal{R}_c(M'_k, D_k) < \sum_k^K \mathcal{R}_c(M_k, D_k)$ when $\mathcal{R}_c$ is strictly monotonically increasing, while

$\sum_{k=1}^{K} \mathcal{R}_c(M'_k, D_k) \leq \sum_{k=1}^{K} \mathcal{R}_c(M_k, D_k)$ when $\mathcal{R}_c$ is monotonically non-decreasing. The proof is completed. □

4.4 The relationship between the ACAFH and CAFH models

In this section, we first briefly review our previously proposed CAFH model, then we show the equivalence between the optimization objectives of the ACAFH and CAFH models.

4.4.1 Our previously proposed CAFH model

In our previous work [19], we first proposed using the heterogeneous hypergraph to concisely model the FCDs in the cost-aware classification problem, then we proposed the CAFH model, which formulated the cost-aware classification problem as an optimization problem that embedded the incident matrix H of the hypergraph. After that, the original discrete objective with nonconvex constraint was relaxed to a solvable continuous optimization objective with an equivalent convex constraint:

$$\begin{aligned} &\min_{M,B\geq \mathbf{0}} \mathcal{L}(\Omega \cdot B) + \lambda \cdot \mathcal{R}_c(M, D) \\ &s.t., M = H \cdot \hat{\Omega} \cdot B, \end{aligned} \tag{9}$$

where $\hat{\Omega} = [A, A] \in \{0, 1\}^{m \times 2m}$ and A is the identity matrix of size $m \times m$. Finally, an ADMM-based nonconvex optimization algorithm was proposed and its convergence property was analyzed.

Our CAFH model was developed to solve the cost-aware classification problem when the features come from limited number of data sources, which typically contains low dimensionality in the dataset. For example, each data source can only generate 11 features and 11 FCCs in Table 1, which can be solved efficiently by our CAFH model. However, in the era of big data, more and more datasets are collected from a large number of data sources, which may potentially contain millions of features and FCCs. For example, when a dataset collected from heterogeneous data sources may contain $m = 10^6$ features and $d = 10^5$ FCCs, the size of its incidence matrix H will be 10^{11} difficult to fit the memory. In addition, the time complexity of the CAFH optimization algorithm is $O(dm^3)$ as we will analyze in Section 5.2, which is over the computation capacity of the contemporary PCs with a single CPU. Although the amount of the feature dependencies can be very large, they barely exist across different data sources. Therefore, the majority costs of memory space and computing resources are redundant and could be optimized.

4.4.2 The equivalence between the CAFH and ACAFH models

It is obvious that the optimization objective of CAFH model in Equation (9) is a special case of ACAFH model's optimization objective in Equation (8), when the CCHH count $K = 1$. Now we prove that the CAFH's objective is still equivalent for any number of CCHHs.

Theorem 2. *The optimization objective of ACAFH model in Equation* (8) *is equivalent to the optimization objective of CAFH model in Equation* (9)*, for any number of CCHHs K, when they choose the same concave regularization term $\mathcal{R}(\cdot)$.*

Proof. Because both optimization objectives are equivalently derived from the original problem objective in Equation (2) except for their respective continuous relaxations, this proof amounts to prove their relaxations are equivalent to each other. Formally, our goal is to prove the relaxation of ACAFH in Equation (7) is equivalent to the relaxation of CAFH:

$$\min_{W,M} \mathcal{L}(W) + \lambda \cdot \mathcal{R}_c(M, D) \; s.t., \; M = H \cdot |W|. \tag{10}$$

Because Equation (7) and Equation (10) share the same loss function and their constraints are equivalent due to $M = \{M_k\}_{k=1}^{K}$, the goal becomes to prove their regularization terms are equivalent. Formally we need to prove the following equation:

$$\lambda \sum_{k}^{K} \mathcal{R}_c(M_k, D_k) = \lambda \mathcal{R}_c(M, D).$$

By Definition 3, the set of CCHHs is actually a unique hypergraph partition of the original hypergraph. Therefore, we have:

$$\begin{aligned} \lambda \sum_{k}^{K} \mathcal{R}_c(M_k, D_k) &= \lambda \sum_{k}^{K} \sum_{i}^{d_k} D_{k,i} \cdot \mathcal{R}(M_{k,i}) && \text{note}^2 \\ &= \lambda \sum_{i}^{d} D_i \cdot \mathcal{R}(M_i) && \text{note}^3 \\ &= \lambda \mathcal{R}_c(M, D) \end{aligned}$$

The proof is completed. □

Algorithm 2 ACAFH Parameter Optimization Algorithm

Require: $K, \{D_k, H_k, \boldsymbol{\alpha}_k, \boldsymbol{\beta}_k\}_{k=1}^{K}$, and η.
Ensure: Solutions of B and M.
Initialize $\rho = 1$, $B, M = \mathbf{0}, \Omega = [\boldsymbol{I}, -\boldsymbol{I}]$.
for $k = 1, \cdots, K$ **do**
  $\hat{\Omega}_k = [\boldsymbol{I}_{\boldsymbol{\beta}_k}, \boldsymbol{I}_{\boldsymbol{\beta}_k}]$
end for
Choose $\varepsilon_p > 0$ and $\varepsilon_d > 0$ // the ADMM algorithm starts here.
repeat
  repeat
    $\nabla = \frac{\partial \mathcal{L}(\Omega \cdot B)}{\partial B}$
    $\hat{\nabla} = \mathbf{0}$
    for $k = 1, \cdots, K$ **do**
      $B_k = B(\boldsymbol{\beta}_k)$
      $\hat{\nabla}_{\boldsymbol{\beta}_k} = (H_k \hat{\Omega}_k)^T H_k \hat{\Omega}_k B_k - H_k \hat{\Omega}_k (M_k + \Gamma_k)$
    end for
    $B \leftarrow \max(B - \eta(\nabla + \hat{\nabla}), \mathbf{0})$
  until Convergence
  for $k = 1, \cdots, K$ **do**
    for $i = 1, \cdots, |\boldsymbol{\beta}_k|$ **do**
      $M_{k,i} \leftarrow$ Equation (15)
    end for
  end for
  Calculate the primal residual p and dual residual d.
  if $r > 10d$ **then**
    $\rho \leftarrow 2\rho$
  else if $10r < d$ **then**
    $\rho \leftarrow \rho/2$
  end if
until $p < \varepsilon_p$ and $d < \varepsilon_d$

5 ACAFH Parameter Optimization

In this section, we first propose an ADMM-based algorithm to optimize the parameters in ACAFH model, which is outlined in Algorithm 2. Then the space and time complexity is analyzed and compared with CAFH.

5.1 ADMM-based algorithm for Optimizing ACAFH

Specifically, we first rewrite the optimization problem in Equation (8) in the augmented Lagrangian form. Then the updates of the target parameters, namely M and B, are elaborated. Finally, the time and space complexity are

2. Sum up all hyperedges' reweighted weights in all CCHHs decomposed from the hypergraph

3. Sum up all hyperedges' reweighted weights in the hypergraph, which is equal to previous step because they are actually share the same set of hyperedges.

analyzed and compared with the algorithm of the CAFH model.

To employ the ADMM framework, the relaxed objective function with the convex constraints in Equation (8) is transformed into the augmented Lagrangian form:

$$L_\rho(M,B)=\mathcal{L}(\Omega B)+\lambda\sum_k^K\|\text{diag}({D_k}^{1/p})M_k\|_p^p+ \frac{\rho}{2}\sum_k^K\left(\|M_k-H_k\hat{\Omega}_kB_k+\Gamma_k\|_F^2-\|\Gamma_k\|_F^2\right) \quad (11)$$

where ρ is the penalty parameter and Γ_k is the dual variable corresponding to the kth constraint. Thus, solving Equation (11) amounts to alternately optimizing the subproblem of B and M, as elaborated in turn below.

5.1.1 Update B

The subproblem of B-update is as follows:

$$\min_{B\geq 0}\mathcal{L}(\Omega\cdot B)+\frac{\rho}{2}\sum_k^K\|M_k-H_k\hat{\Omega}_kB_k+\Gamma_k\|_F^2 \quad (12)$$

Equation (12) can be solved by the backtracking Armijo line search [2] method:

$$B\leftarrow \text{prox}_{\geq 0}(B-\eta(\nabla+\hat{\nabla})) \quad (13)$$

where $\{\nabla,\hat{\nabla}\}\in\mathbb{R}^{2m\times 1}$ such that $\nabla=\frac{\partial\mathcal{L}(\Omega\cdot B)}{\partial B}$ and $\hat{\nabla}=\{\hat{\nabla}_{\boldsymbol{\beta_k}}\}_k^K$ where $\hat{\nabla}_{\boldsymbol{\beta}_k}=(H_k\hat{\Omega}_k)^TH_k\hat{\Omega}_kB_k-H_k\hat{\Omega}_k(M_k+\Gamma_k)$

5.1.2 Update M

The subproblem of M-update is as follows:

$$M_k\leftarrow\min_{M_k\geq 0}\lambda\|\text{diag}(D_k^{1/p})M_k\|_p^p+\frac{\rho}{2}\|M_k-H_k\hat{\Omega}_kB_k+\Gamma_k\|_F^2 \quad (14)$$

where $k=1,\ldots,K$, which are all separable and each subproblem of $M_{k,i}$ is:

$$\min_{M_{k,i}\geq 0}h_k(M_{k,i})=\lambda D_{k,i}M_{k,i}^p+\frac{\rho}{2}(M_{k,i}-H_{k,i}\hat{\Omega}_kB_k+\Gamma_{k,i})^2 \quad (15)$$

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 $M_{k,j}^{1/2}=x$, the derivative of Equation (15) can be transformed to the following:

$$x^3-(H_{k,i}\hat{\Omega}_kB_k-\Gamma_{k,i})x+\frac{\lambda}{2\rho}D_{k,i}=0 \quad (16)$$

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 [35]. $\mathbb{C}$ denotes the complex value domain. Therefore, the analytical solution to $M_{k,i}$ is:

$$M_{k,i}^*=\max(\max(x_r^*),0)^2,\text{where } x_r^*=\{s|s\in\mathbb{R},s\in x^*\} \quad (17)$$

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

$$x^4-\rho(H_{k,i}\hat{\Omega}_kB_k+\Gamma_{k,i})x+\lambda/(2\rho)D_{k,i}=0 \quad (18)$$

Therefore, we have:

$$M_{k,i}^*=\begin{cases}\max(\max(x_r^*),0)^3, & \text{when } x_r^*\neq\emptyset\\ 0, & \text{when } x_r^*=\emptyset\end{cases} \quad (19)$$

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

5.2 Algorithm Implementation and Complexity Analysis

The algorithm is initialized as the case for $p=1$, which is a convex problem and can provide a good initial guess. We implement the proposed ACAFH model in Matlab [4].

Algorithm 2 takes $\{H_k\}_{k=1}^K$ as the input instead of the entire H. So the space complexity of the Algorithm 2 is only $O(\sum_k^K m_kd_k)\approx O(K\overline{m}\overline{d})$, which is less than $O(dm)\approx O(K^2\overline{m}\overline{d})$ in CAFH model, where $\overline{m}$ and $\overline{d}$ are respectively the average number of vertices and hyperedges in each connected component. As the ADMM-based algorithm typically can provide the results that are good enough for prediction in several dozens of iterations, the outer loop in Lines 6-27 as a constant number of iterations. The number of iterations of updating B Lines 7-15 can typically be a few thousands, which depends on the choice of η for both ACAFH and CAFH optimization algorithms. For each iteration of updating B, the time complexity is $O(\sum_k^K d_k\cdot m_k^3)$ that is dominated by the time complexity of computing the gradients in Line 12, which also absorbs the time complexity of updating M. In contrast, in CAFH optimization algorithm, the time complexity of computing the gradients is $O(d\cdot m^3)\approx O(K^4\overline{d}\overline{m}^3)$ that is much larger than $O(\sum_k^K d_k\cdot m_k^3)\approx O(K\overline{d}\overline{m}^3)$ in ACAFH when the hypergraph is highly sparse (i.e., K is large).

6 Experiments

In this section, we first experimental setup is introduced, then we evaluate the effectiveness of the proposed ACAFH model. Then, the efficiency of optimizing the proposed models is explored on various settings.

6.1 Experimental Settings

In this section, the datasets, evaluation metrics, and the comparison methods are introduced in turn. All the experiments were implemented Matlab, and conducted on a 64-bit machine with 16.0 GB memory.

6.1.1 Datasets

• Synthetic datasets

In effectiveness experiments, eight synthetic datasets, each has 1000 samples, were generated with different settings. The generation procedures are 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 $W_i=a_i\cdot b_i$. Here each a_i is sampled from a Gaussian distribution with a mean of zero and a variance of one while b_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, 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 Bernoulli distribution with successful probability ranging from "0.1" to "0.8". This method was used to generate eight synthetic datasets with different sparsities of the H matrix, reflecting

4. The codes are available at https://github.com/qingzheli/ACAFH

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.In efficiency experiments, the related synthetic datasets are generated with various feature counts and sparsity levels accordingly while the generation process is the same as described here.

• 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 Fig. 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.

• Metrics

For effectiveness experiments, the F1 score, namely $F1 = 2 \cdot \text{precision} \cdot \text{recall}/(\text{precision} + \text{recall})$, is utilized as the major metric. The other metrics include: 1) prediction runtime, which represents the total amount of time spent on prediction including both feature generation and the model prediction using the generated features. 2) the number of selected features, which is the summation of the count of the non-zero entries in feature weights W. 3). the number of selected feature computational components, which is the summation of the count of the non-zero entries in $H \cdot W$. The 5-fold cross-validation is performed, and the average F1 scores and prediction time are recorded.

For efficiency experiments, the CPU time of training the models and the peak memory usages are utilized for efficiency evaluation.

• Competing Methods

We compare the proposed model with 5 prediction time-sensitive classifiers and 2 generic classifiers: ℓ_1-regularized logistic regression [27], [36], re-weighted-ℓ_1-regularized logistic regression [35], Cost-Sensitive Tree of Classifiers (CSTC) [33], Greedy Miser [28], Directed Acyclic Graph for cost-constrained prediction (DAG) [37], Neural Network, and Random Forests [38]. These methods 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 λ to regularize the feature weights (e.g., $\lambda\|W\|_1$). The larger the value of λ is, the fewer the selected features will be.

Reweighted-ℓ_1-regularized logistic regression (reweighted L1). This is a generalized version of L1. Here the respective cost of each feature 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 λ on its regularization term (e.g., $\lambda\|H^T D \circ W\|_1$) to balance the empirical loss and time cost, where $H^T D$ provides the time costs for each feature and "$\circ$" denotes the Hadamard product to reweight the features' weights in the regularization term.

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.

Greedy Miser. 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.

Neural Network and Random Forest. These two methods do not consider the prediction time at all. But they are considered as one of the state-of-the-art general-purpose classifiers. The fully connected shallow neural network with three layers implemented in Matlab deep learning toolbox and the standard random forest with sixty bags implemented in Matlab statistics and machine learning toolbox are employed in our experiments.

Cost-Aware classification using FCD Heterogeneous hypergraph (CAFH) Our previously proposed methods [19]. 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.

6.2 Effectiveness Evaluation

In this section, we first evaluate the effectiveness of ours and all comparison methods, then we analyze the impact

of feature computational dependency on prediction runtime and prediction F1 scores.

6.2.1 Performance on the Effectiveness

The performance of the proposed and comparison methods are illustrated and discussed on synthetic datasets and real-world datasets in turn.

• **Performance on Synthetic Datasets**

Fig. 2 shows the F1-runtime performance histograms for all the methods on the 8 synthetic datasets with increasing prediction runtime budget levels. The bars represent the highest F1 scores within the corresponding runtime budget. Some methods' bars are missing from lower levels of runtime (e.g., runtime less than 200 microseconds). Because none of the runtimes satisfies the given time budget after the trade-off parameters were intensively tuned. For neural network and random forests that do not consider the prediction time, all features are selected, so their bars are only shown in the highest level of runtime budget (i.e., runtime is greater than 300 microseconds). It is obvious that less prediction runtime budget with higher F1 scores are desired. Our currently proposed ACAFH models and our previously proposed CAFH models [19] consistently outperforms the other methods on most budget levels. As seen in Fig. 2, on < 200 microseconds level, none of the methods perform well because the F1 score is 0.67 when simply predicting all samples as positive. On < 250 microseconds level, our outperform other comparison methods on six out of eight datasets. On < 300 microseconds level, our methods achieve F1 scores over 0.9 on all eight datasets while the best comparison method DAG on this runtime level only achieves about F1 scores of 0.85. On runtime > 300 microseconds level, the neural network, the L1 and reweighted L1 methods all achieve competitive F1 scores with our methods, but our methods can achieve the similar results within a lower runtime budget. The F1 scores of the CAFH (p=1/2) method are similar to ACAFH (p=1/2) method since these two models are theoretically equivalent to each other. Although the performance of the proposed ACAFH model is very close to the previously proposed CAFH model, as we will show in Section 6.3, their training time and memory cost are very different when the features are from different data sources. The F1 scores of the CAFH/ACAFH (p=2/3) methods perform slightly worse than the CAFH/ACAFH (p=1/2) methods be-cause $\ell_{1/2}$ quasi-norm can provides a better approximation of $\mathbf{I}(\cdot)$ than $\ell_{2/3}$ quasi-norm.

• **Performance on Real-world Datasets**

Fig. 3 demonstrates the effectiveness of our proposed ACAFH methods compared with the other methods on all six real-world datasets. Our methods achieve an average F1 score of over 0.99 within 1 microsecond prediction runtime on all six real-world datasets. On runtime budget < 1 microsecond level, the reweighted L1 method generally performs well than other comparison methods, but still worse than our methods in Fig. 3 (b) directed DBPOWER, (c) directed DJI Spark, and (e) undirected DBPOWER datasets. The random forest method achieves the highest F1 scores on all six datasets. However, its prediction runtimes are much longer than other methods since the random forest method typically needs lots of features in different trees. DAG tends to achieve high F1 scores within 2 microseconds on the last three datasets, which only contain 18 features. For the first three datasets with 54 features, the DAG method fails to select the optimal set of features to reduce the prediction runtime.

6.2.2 Analysis of Feature Computational Dependencies

This section analyzes the effectiveness of the selected features in reducing runtime and retaining F1 score. The prediction runtime versus the number of selected features and the F1 score versus the number of selected features are empirically analyzed on both synthetic and real-world datasets.

For synthetic datasets, only two synthetic datasets are plotted to show the trends due to space limitations. As shown in the first row of Fig. 4, the dashed lines corresponding to the L1 and reweighted L1 methods are mostly above the dotted lines corresponding to our methods, which clearly illustrates the proposed methods use less prediction runtime, when selecting the same number of features, by selecting the optimal set of feature computational components through FCD. On the other hand, as shown in the second row of Fig. 4, our methods plotted in dotted lines can still achieve competitive F1 scores of the L1 and reweighted L1 methods. Because our methods consider both prediction runtime and prediction error, in contrast, the L1 and reweighted L1 methods only consider minimizing the prediction error.

For real-world datasets, the results shown in Fig. 5 are consistent with Fig. 4 for the same reason. Moreover, as shown in the 2nd-row of Fig. 5, which plots the curves between the number of Feature Computational Components (FCCs) versus the number of the selected features, our methods tend to select more features with less number of FCCs than the comparison methods. The curves of our methods in the 1st-row figures are highly correlated to the curves in the 2nd-row figures, which can empirically explain why our methods need less prediction runtime than the comparison methods.

6.3 Efficiency Evaluation

This section evaluates the efficiency of the proposed ACAFH model in the training phase. Unlike the identical performance on the effectiveness, the CAFH and ACAFH perform quite differently on their efficiency in term of the training/optimization time. When the total feature count is high (i.e., m is large), or when the feature dependencies are rare (i.e., H is sparse), the training time and the memory usage of the ACAFH model are much less than CAFH.

6.3.1 Training Time vs. Number of total features

• **Training time on synthetic datasets:** In this experiment, The synthetic datasets used in this section are generated in the way as described in Section 6.1.1. Six synthetic datasets were generated which contain 10000 training samples each, $m \in \{1000, 2000, \cdots, 6000\}$ features and $\Psi = 0.0001$, which controls the sparsity level of the incidence matrix H, is fixed. Fig. 6 (a) and Fig. 6 (b) compare the training time of CAFH and ACAFH for $p = 1/2$ and $p = 2/3$ cases, when the number of features m are varying from 1000 to 6000. As shown, the training time increases with the growing of the total feature count for both $p = 1/2$ and $p = 2/3$ cases. However, ACAFH increases much slower than CAFH. This is because the computational cost of ACAFH grows slowly after decomposing the sparse hypergraph to several connected components as discussed in Section 5.1.

• **Training time on real-world datasets:** In this experiment,

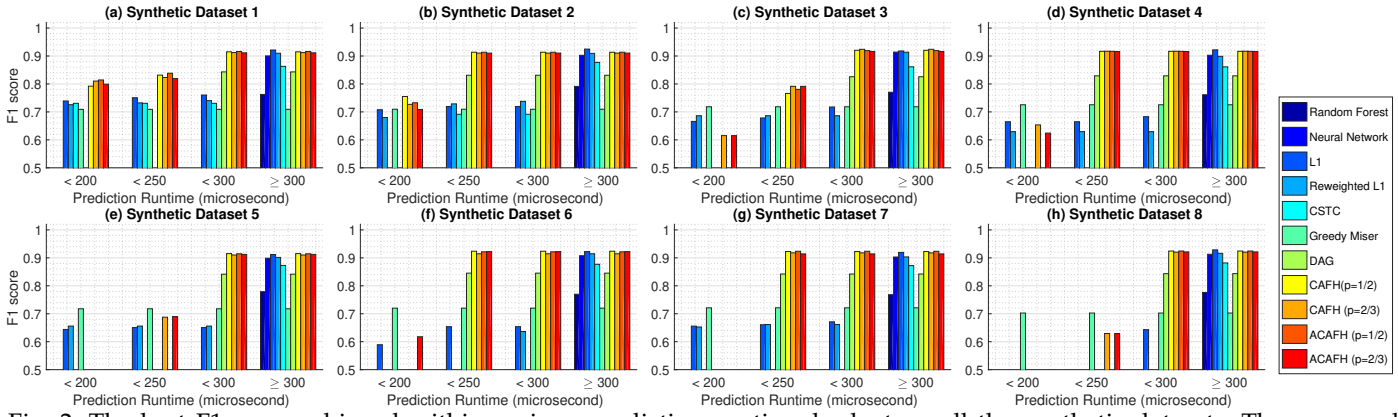


Fig. 2: The best F1 score achieved within a given prediction runtime budget on all the synthetic datasets. The proposed ACAFH models and CAFH models outperform other comparison methods on lower budget levels.

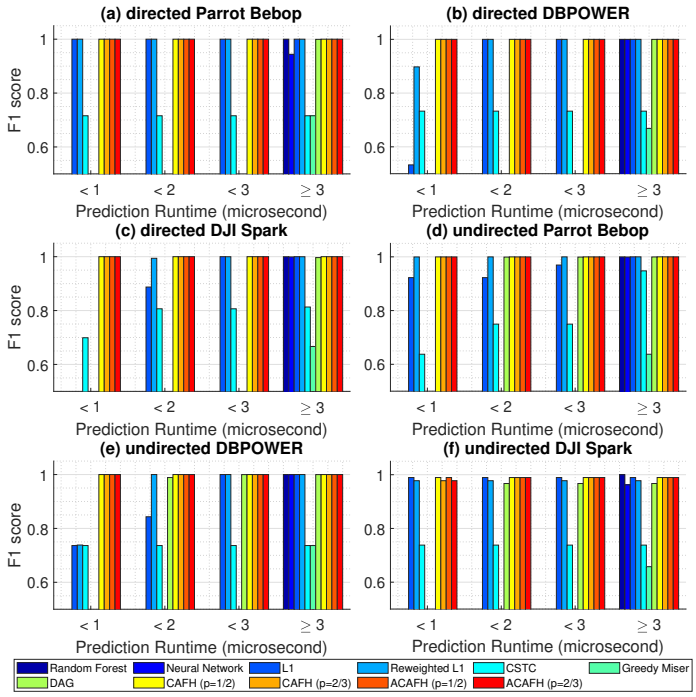


Fig. 3: The best F1 score achieved within a given prediction runtime budget on all the real-world datasets. The proposed ACAFH and CAFH models achieve over 0.99 average F1 scores within 1 microsecond on all six real-world datasets.

the training time of the proposed ACAFH model and other comparison methods are evaluated. All methods are implemented in Matlab, except for Greedy Miser, which is compiled to C++ from Matlab by its author. To illustrate the training time with various numbers of features, we create five additional datasets, which contain 1080, 2160, 3240, 4320, and 5400 features, by repeating the original 54 features 20, 40, 60, 80, and 100 times from the real-world directed Parrot Bebop dataset. Recall the original 54 features come from 2 sources $\times$ 3 direction flow $=$ 6 data sources, and there are 5 Connected Components in Heterogeneous Hypergraph (CCHH) for each data source. Therefore, the original directed Parrot Bebop dataset contains 5 CCHHs $\times$ 6 data sources $=$ 30 CCHHs (i.e., $K = 30$). Naturally, the remaining five datasets contain $K = 600, K = 1200, K = 1800, K = 2400, K = 3000$ CCHHs. The training time versus the number of features of all methods is shown in

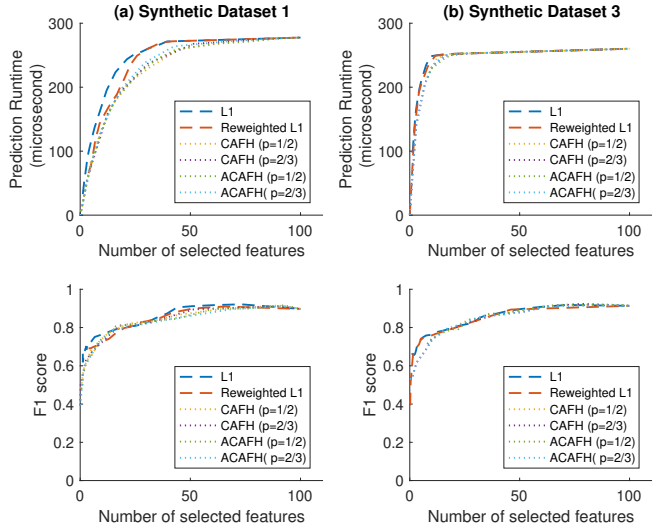


Fig. 4: The prediction runtime (1st-row) and the F1 score (2nd-row) vs. number of selected features on synthetic datasets. When selecting the same number of features, the proposed methods (plotted in dotted lines) take much less time than the L1 and reweighted L1 methods (plotted in dashed lines), meanwhile, the proposed methods can still achieve competitive F1 scores as the comparison methods.

Fig. 7. All methods can be trained on all the datasets in less than 1500 seconds except the DAG method, whose training time increases very fast with the growth of the number of features. As seen in Fig. 7, the training time of the proposed ACAFH model (plotted solid lines) increases much slower than our previous CAFH model (plotted in dashed lines) as the time complexity versus analyzed in Section 5.2. For other comparison methods, the L1 and reweighted L1 methods grow linearly, but their growth rates are very high such that these two methods cost more training time than the proposed methods. The shallow neural network, random forest, greedy miser, and CSTC methods all take slightly less time than the proposed methods. However, their performances on effectiveness are much worse than the proposed methods, as shown in the previous section.

6.3.2 *Peak memory usage*

• **Memory usage on synthetic datasets:** In this experiment, two groups of datasets were generated with 5 synthetic datasets in each group. The sparsity level Ψ was 10^{-4} in the first group, while the sparsity level Ψ was 10^{-5} in the second

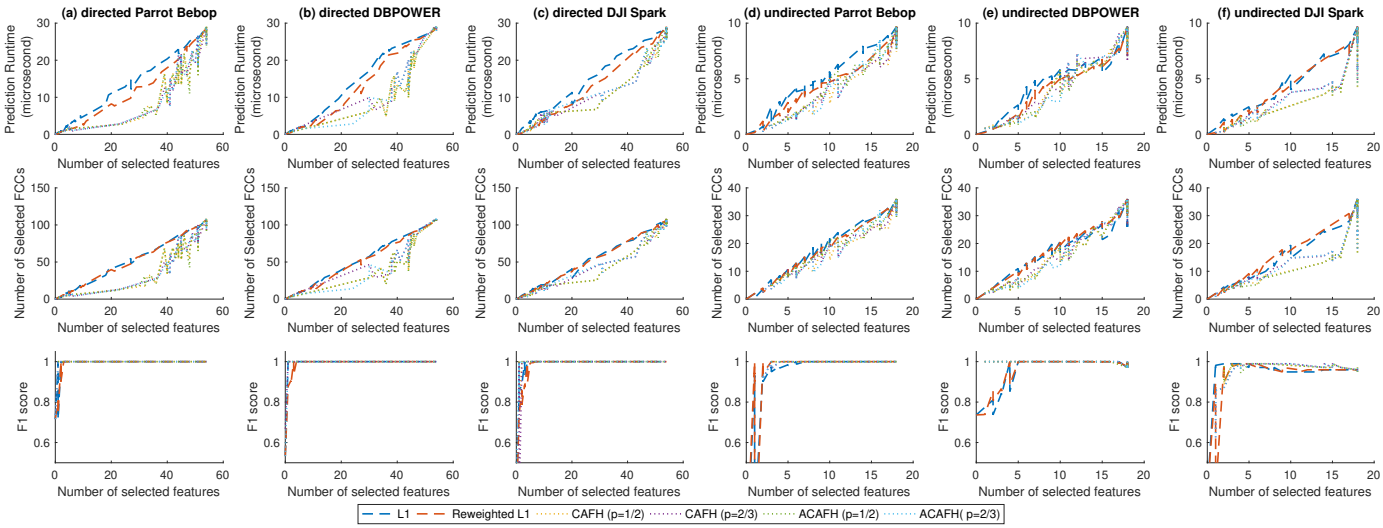


Fig. 5: The prediction runtime (1st-row), the number of selected FCCs (2nd-row), and the F1 score (3rd-row) vs. number of selected features on all real-world datasets. When selecting the same number of features, the proposed methods (plotted in dotted lines) take much less time than the L1 and reweighted L1 methods (plotted in dashed lines in the 1st-row figures) by selecting less number of FCCs (as shown in the 2nd-row figures), meanwhile, the proposed methods can achieve even better F1 scores than the comparison methods.

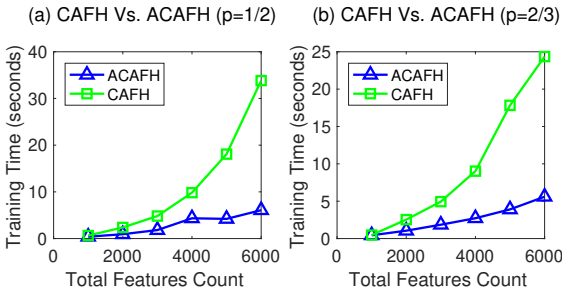


Fig. 6: Training time on synthetic datasets with various feature counts

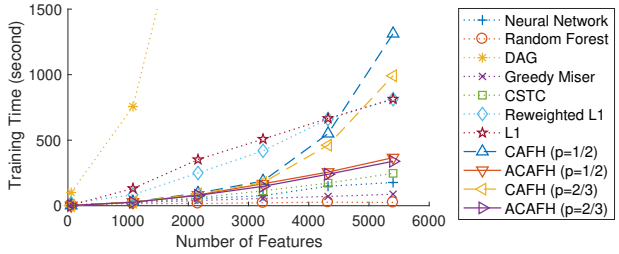


Fig. 7: Training time on real-world Dataset with various feature counts

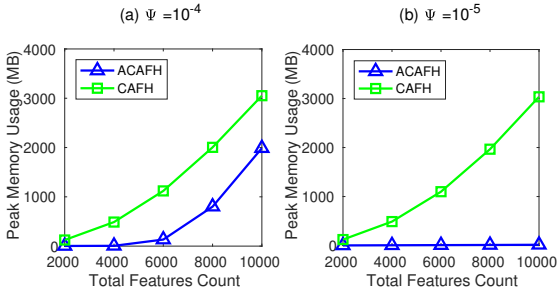


Fig. 8: The peak memory usages of CAFH and ACAFH with various feature counts on two sparsity levels

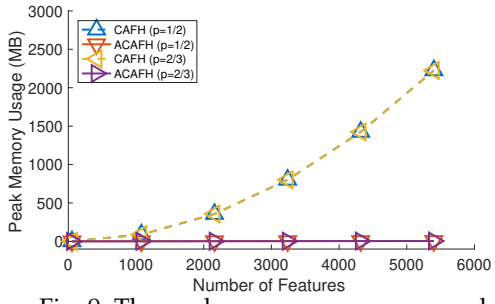


Fig. 9: The peak memory usages on real-world dataset

group. For each group, we recorded the peak memory usages during the training phase with various feature counts ranging from 2000 to 10000. As seen in Fig. 8, the optimization of ACAFH takes much less memory than CAFH when the incidence matrix H is sparse (e.g. $\Psi \leq 10^{-5}$). In addition, when $\Psi = 10^{-4}$ the peak memory usage of ACAFH model grows faster than the peak memory usage when $\Psi = 10^{-5}$. This is interesting because the space complexity is related to the size of the largest CCHH and the size of the largest CCHH can still be very large when H is not sparse enough. When the H is sparse enough, the size of the largest CCHH may grow very slowly with the growth of the feature counts, so in this case, memory usage of ACAFH is quite insensitive to the feature count. In contrast, the memory usage of CAFH grows much faster than the ACAFH model as analyzed in Section 5.2. When H is dense, the size of the largest CCHH can be close to the original Hypergraph, in this case, the memory usage of the ACAFH model is similar to the CAFH model.

• **Memory usage on real-world datasets:** In this experiment, the same real-world datasets described in previous training time experiments are used. The size of raw data also increases with the growing number of features, which uses the same memory for all methods. To properly show the memory usage by different algorithms, we just plotted the memory usage by the algorithm in Fig. 9 by excluding the memory used by the raw data. Because other comparison methods except for our previously proposed CAFH model do not consider the hypergraph to model the feature computational dependencies, which uses the most memories, we

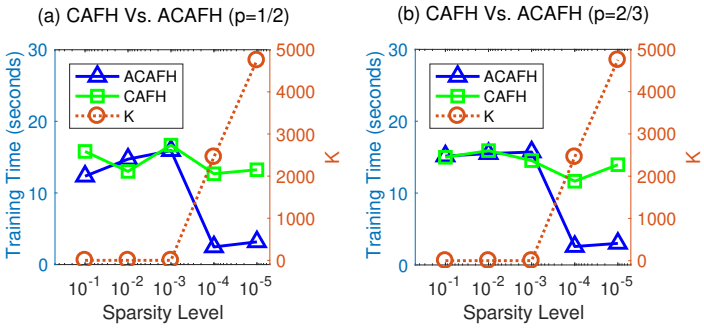


Fig. 10: Training time with various sparsity levels

only compare the memory usage with the CAFH models in this experiment. As seen in Fig. 9, with the growth of the number of features, the optimization algorithm of the CAFH model plotted in dashed lines grows quadratically, in contrast, the proposed optimization algorithm of the ACAFH model plotted in solid lines grows linearly with a very small growth rate.

6.3.3 *Training Time vs. Sparsity Level*

In this experiment, five synthetic datasets with 5000 features with 10000 training samples were generated by varying the value of Ψ from 10^{-1} to 10^{-5}. Fig. 10 (a) and Fig. 10 (b), which respectively correspond to the $p = 1/2$ and $p = 2/3$ cases, compare the training time and the counts of connected components K versus various sparsity levels. As shown, when the incidence matrix H is non-sparse (i.e. $\Psi \geq 10^{-3}$), the number of connected components K remains closely to 1. Therefore, CAFH and ACAFH take similar time on training. However, when the sparsity level H increases (i.e. $\Psi < 10^{-3}$), the number of connected components K quickly grows. Meanwhile, the training time of ACAFH is much less than CAFH, which validated our analysis in Section 5.2.

7 Conclusion and Future Work

In this paper, we proposed an Accelerate Cost-aware classification for large-scale and sparse feature computational dependencies heterogeneous hypergraph model to effectively reduce the prediction-time cost and retain the model accuracy. Specifically, we first model the feature computational dependency (FCD) as FCD heterogeneous hypergraph and propose a concise objective function with a faithful representation of the runtime costs. To optimize this objective function efficiently with the growth feature counts coming from multiple data sources, we decompose the original FCD into a set of Connected Components in Heterogeneous Hypergraphs (CCHHs). By using the CCHHs in stead of previous FCD hypergrah, we developed an Accelerated Cost-Aware classification for large-scale and sparse FCD Heterogeneous hypergraph (ACAFH) model, which is equivalent to the CAFH mode but could be trained much faster. Extensive experiments on several synthetic and real-world datasets demonstrated the advantageous performance of the proposed models over the existing methods for cost-sensitive classification. Detailed analysis on the selected features and FCCs were also presented to show the competing performance on the effectiveness of the proposed method. Finally, the time and memory costs of training the proposed ACAFH model were tested on several large-scale synthetic and real-world datasets, which demonstrated the improvements of the proposed ACAFH model comparing to previously proposed CAFH model.

For future work, the space and time complexity of the proposed ACAFH optimization algorithm actually depends on the size of the largest CCHH. As shown in Fig. 8 (a) and Fig. 10, when the sparsity level is low, the size of the largest CCHH could be still very large such that the memory and time costs will be high during the training phase. One possible direction could be breaking large CCHHs into smaller CCHHs by ignoring some FCDs from original FCDs. However, such decomposition is lossy, which will cause the suboptimal solution of the ACAFH model. Another promising direction could be applying the proposed framework with deep models. Currently, the proposed ACAFH models apply the framework to the logistic regression model. In the future, we believe it will be interesting to explore the application on deep neural networks to reduce the number of features as well as the size of the neural network model.

Acknowledgments

This work was supported by This work was supported by the National Science Foundation grant: #1841520, #1755850, #1907805, Jeffress Trust Awards, and NVIDIA GPU Grant, Y. Ye's work is partially supported by the NSF CNS-1946327, CNS-1814825, CNS-1940859, III-1951504, and OAC-1940855, the DoJ/NIJ 2018-75-CX-0032.

References

[1] B. Poblete, J. Guzmán, J. Maldonado, and F. Tobar, "Robust detection of extreme events using twitter: Worldwide earthquake monitoring," *IEEE Transactions on Multimedia*, no. 10, 2018.

[2] L. Zhao, Q. Sun, J. Ye, F. Chen, C.-T. Lu, and N. Ramakrishnan, "Multi-task learning for spatio-temporal event forecasting," in *Proceedings of the 21th ACM SIGKDD*, 2015.

[3] J. Wehrmann, G. S. Simões, R. C. Barros, and V. F. Cavalcante, "Adult content detection in videos with convolutional and recurrent neural networks," *Neurocomputing*, 2018.

[4] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, no. 3, Jun. 2017.

[5] P. Teisseyre, D. Zufferey, and M. Słomka, "Cost-sensitive classifier chains: Selecting low-cost features in multi-label classification," *Pattern Recognition*, 2019.

[6] Q. Xu and R. Zheng, "When data acquisition meets data analytics: A distributed active learning framework for optimal budgeted mobile crowdsensing," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 2017, pp. 1–9.

[7] L. Li, U. Topkara, and N. Memon, "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*, 2011.

[8] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *Proceedings of the 23rd ACM SIGKDD*, New York, NY, USA, 2017.

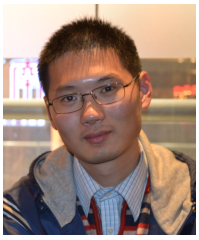
[9] Y. Fan, Y. Ye, and L. Chen, "Malicious sequential pattern mining for automatic malware detection," *Expert Systems with Applications*, 2016.

[10] L. Chen, S. Hou, and Y. Ye, "Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks," in *Proceedings of the 33rd ACSAC*, New York, NY, USA, 2017.

[11] J. Wang and L. Zhao, "Multi-instance domain adaptation for vaccine adverse event detection," in *Proceedings of WWW 2018*, Republic and Canton of Geneva, Switzerland, 2018.

[12] J. Pang, H. Lin, L. Su, C. Zhang, W. Zhang, L. Duan, Q. Huang, and B. Yin, "Accelerate convolutional neural networks for binary classification via cascading cost-sensitive feature," in *Image Processing (ICIP)*, 2016.

[13] L. Reyzin, "Boosting on a budget: Sampling for feature-efficient prediction," in *Proceedings of the 28th ICML (ICML-11)*, 2011.
[14] M. Tan and J. C. Schlimmer, "Cost-sensitive concept learning of sensor use in approach and recognition," in *Proceedings of the sixth international workshop on Machine learning*, 1989.
[15] A. Grubb and D. Bagnell, "Speedboost: Anytime prediction with uniform near-optimality," in *Artificial Intelligence and Statistics*, 2012.
[16] C.-L. Huang and C.-J. Wang, "A ga-based feature selection and parameters optimizationfor support vector machines," *Expert Systems with applications*, no. 2, 2006.
[17] M. J. Kusner, W. Chen, Q. Zhou, Z. E. Xu, K. Q. Weinberger, and Y. Chen, "Feature-cost sensitive learning with submodular trees of classifiers." in *AAAI*, 2014.
[18] L. Zhao, J. Ye, F. Chen, C.-T. Lu, and N. Ramakrishnan, "Hierarchical incomplete multi-source feature learning for spatiotemporal event forecasting," in *Proceedings of the 22Nd ACM SIGKDD*, New York, NY, USA, 2016.
[19] L. Zhao, A. Alipour-Fanid, M. Slawski, and K. Zeng, "Prediction-time efficient classification using feature computational dependencies," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2787–2796.
[20] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.
[21] J. Wang, F. Yu, X. Chen, and L. Zhao, "Admm for efficient deep learning with global convergence," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: ACM, 2019, pp. 111–119. [Online]. Available: http://doi.acm.org/10.1145/3292500.3330936
[22] *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019. [Online]. Available: https://www.aaai.org/Library/AAAI/aaai19contents.php
[23] J. Pujara, H. Daumé III, and L. Getoor, "Using classifier cascades for scalable e-mail classification," in *Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference*, 2011.
[24] X. Guo, A. Alipour-Fanid, L. Wu, H. Purohit, X. Chen, K. Zeng, and L. Zhao, "Multi-stage deep classifier cascades for open world recognition," in *Proceedings of the ACM CIKM*, 2019.
[25] L. Li, U. Topkara, B. Coskun, and N. Memon, "Cocost: a computational cost efficient classifier," in *ICDM'09*, 2009.
[26] C. Ferri, P. Flach, and J. Hernández-Orallo, "Learning decision trees using the area under the roc curve," in *ICML*, 2002.
[27] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani *et al.*, "Least angle regression," *The Annals of statistics*, no. 2, 2004.
[28] Z. Xu, K. Q. Weinberger, and O. Chapelle, "The greedy miser: learning under test-time budgets," in *Proceedings of the 29th ICML*, 2012.
[29] A. Kapoor and R. Greiner, "Learning and classifying under hard budgets," in *ECML*, 2005.
[30] K. Deng, Y. Zheng, C. Bourke, S. Scott, and J. Masciale, "New algorithms for budgeted learning," *Machine learning*, no. 1, 2013.
[31] F. Nan and V. Saligrama, "Adaptive classification for prediction under a budget," in *NIPS*, 2017.
[32] C. X. Ling, V. S. Sheng, and Q. Yang, "Test strategies for cost-sensitive decision trees," *IEEE Transactions on Knowledge and Data Engineering*, no. 8, 2006.
[33] Z. E. Xu, M. J. Kusner, K. Q. Weinberger, M. Chen, and O. Chapelle, "Classifier cascades and trees for minimizing feature evaluation cost." *Journal of Machine Learning Research*, no. 1, 2014.
[34] R. Chartrand and W. Yin, "Nonconvex sparse regularization and splitting algorithms," in *Splitting Methods in Communication, Imaging, Science, and Engineering*, 2016.
[35] A. Y. Aravkin, J. V. Burke, and G. Pillonetto, "Sparse/robust estimation and kalman smoothing with nonsmooth log-concave densities: Modeling, computation, and theory," *The Journal of Machine Learning Research*, no. 1, 2013.
[36] L. Zhao, Q. Sun, J. Ye, F. Chen, C.-T. Lu, and N. Ramakrishnan, "Multi-task learning for spatio-temporal event forecasting," in *Proceedings of the 21th ACM SIGKDD*, New York, NY, USA, 2015.
[37] J. Wang, K. Trapeznikov, and V. Saligrama, "Efficient learning by directed acyclic graph for resource constrained prediction," in *NIPS*, 2015.
[38] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.



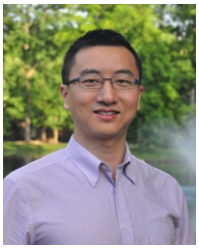
Qingzhe Li is a Ph.D. candidate in information technology supervised by Dr. Liang Zhao. He received his Master's degree in computer science from State University of New York at New Paltz advised by Dr. Keqin Li. His research interests include time series and spatiotemporal data mining, sparse feature learning, and deep learning on graphs.

Amir Alipour-Fanid is a Ph.D. student in electrical and computer engineering department supervised by Dr. Kai Zeng. His research focus is on the active safety application of vehicle to vehicle communications from the Cyber-Physical Systems (CPS) pesrspective.

Martin Slawski is an Assistant Professor in the Department of Statistics at George Mason University, USA. He received his Ph.D. degree from in the Machine Learning Group at Saarland University, Germany. His research interests include Dimension reduction, Data Compression, High-Dimensional Statistics, Convex Geometry, Discrete and Continuous Optimization, Matrix Factorizations, Non-asymptotic Random Matrix Theory, Applications in the Life Sciences.

Yanfang Ye is Leonard Case Jr. Associate Professor at Case Western Reserve University. She received her Ph.D. in computer science from Xiamen University. She was an assistant professor and then associate professor in the department of computer science and electrical engineering (CSEE) at West Virginia University. Her research areas mainly include Data Mining, Machine Learning, Cybersecurity, and Health Intelligence.

Lingfei Wu is a Research Staff Member in the IBM AI Foundations Labs, Reasoning group at IBM T. J. Watson Research Center. He earned his Ph.D. degree in computer science from College of William and Mary in 2016. His research interests mainly span in Machine Learning, Deep Learning, Representation Learning, Natural Language Processing, and Numerical Linear Algebra.



Kai Zeng is an associate professor in the Department of Electrical and Computer Engineering at George Mason University. He is also affiliated with the Department of Computer Science and the Center for Secure Information Systems. He received his Ph.D. degree in Electrical and Computer Engineering at Worcester Polytechnic Institute. His research interest include Cybersecurity, Cyber-Physical Systems, Physical Layer Security, Cognitive Radio Networks, Network Forensics.

Liang Zhao is an Assistant Professor at Information Science and Technology Department of George Mason University, USA. He received the Ph.D. degree from Virginia Tech, USA. His research interests include interpretable machine learning, societal event forecasting, sparse feature learning, social media mining, nonconvex optimization, and deep learning on graphs.