8-2017

Large-scale online feature selection for ultra-high dimensional sparse data

Yue WU
University of Science and Technology of China

Steven C. H. HOI
Singapore Management University, chhoi@smu.edu.sg

Tao MEI
University of Science and Technology of China

Nenghai YU
University of Science and Technology of China

Published in ACM Transactions on Knowledge Discovery from Data,
2017 June, Volume 11, Issue 4
http://doi.org/10.1145/3070646

Large-Scale Online Feature Selection for Ultra-High Dimensional Sparse Data

YUE WU, University of Science and Technology of China, Singapore Management University
STEVEN C. H. HOI, Singapore Management University
TAO MEI, University of Science and Technology of China, Microsoft Research Asia
NENGHAI YU, University of Science and Technology of China

Feature selection (FS) is an important technique in machine learning and data mining, especially for large-scale high-dimensional data. Most existing studies have been restricted to batch learning, which is often inefficient and poorly scalable when handling big data in real world. As real data may arrive sequentially and continuously, batch learning has to retrain the model for the new coming data, which is very computationally intensive. Online feature selection (OFS) is a promising new paradigm that is more efficient and scalable than batch learning algorithms. However, existing online algorithms usually fall short in their inferior efficacy. In this article, we present a novel second-order OFS algorithm that is simple yet effective, very fast and extremely scalable to deal with large-scale ultra-high dimensional sparse data streams. The basic idea is to exploit the second-order information to choose the subset of important features with high confidence weights. Unlike existing OFS methods that often suffer from extra high computational cost, we devise a novel algorithm with a MaxHeap-based approach, which is not only more effective than the existing first-order algorithms, but also significantly more efficient and scalable. Our extensive experiments validated that the proposed technique achieves highly competitive accuracy as compared with state-of-the-art batch FS methods, meanwhile it consumes significantly less computational cost that is orders of magnitude lower. Impressively, on a billion-scale synthetic dataset (1-billion dimensions, 1-billion non-zero features, and 1-million samples), the proposed algorithm takes less than 3 minutes to run on a single PC.

CCS Concepts: • **Computing methodologies** → **Feature selection**; *Supervised learning by classification;*

Additional Key Words and Phrases: Feature selection, sparsity, second-order online learning, ultra-high dimensionality

1. INTRODUCTION

In machine learning and data mining, feature selection (FS) is the process of removing irrelevant and redundant features from data toward model construction. It is a very

This research is supported by the National Research Foundation, Prime Ministers Office, Singapore under its International Research Centres in Singapore Funding Initiative, the National Natural Science Foundation of China (No. 61371192) and the Key Laboratory Foundation of the Chinese Academy of Sciences (CXJJ-17S044). This work was done when the first author visited Dr. Hoi's research group.
Authors' addresses: Y. Wu and N. Yu, Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei 230027, China; emails: wye@mail.ustc.edu.cn, ynh@ustc.edu.cn; S. C. H. Hoi (corresponding author), School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902, Singapore; email: chhoi@smu.edu.sg; T. Mei, Microsoft Research Asia, Beijing 100080, China; email: tmei@microsoft.com.

important technique in the era of big data today. It has found applications in a wide range of domains, particularly for scenarios with high-dimensional data [Bolón-Canedo et al. 2015; Zhai et al. 2014]. FS has been extensively studied in which various algorithms have been proposed [Hua et al. 2009; Liu and Yu 2005].

Despite the extensive research efforts in literature, most FS methods are restricted to batch learning settings [Bolón-Canedo et al. 2013]. However, batch learning has critical drawbacks for big data applications. One drawback is that they require the entire training dataset to be loaded into memory. This is obviously non-scalable when solving real-world applications with large-scale datasets that exceed the memory capacity. Another drawback is that batch learning methods usually assume all training data and their full set of features have been made available prior to the learning process. This assumption does not always hold in many real-world applications where data arrive sequentially (e.g., internet data) or novel features may appear incrementally (e.g., spam email filtering). These drawbacks make traditional batch FS techniques non-practical for emerging big data applications.

To overcome the drawbacks of batch FS, online feature selection (OFS) has been explored recently [Hoi et al. 2012; Wang et al. 2014; Wu et al. 2010; Yang et al. 2013]. One state-of-the-art scheme in Wang et al. [2014] attempts to resolve FS by exploring online learning techniques. Although it is far more efficient and scalable than batch FS techniques, it still falls short in requiring linear time complexity with respect to feature dimensionality. Besides, the learning accuracy is often not as well as batch FS algorithms.

In this article, we argue that existing solutions are still not feasible due to the high computation and memory cost in real-world applications. We propose a simple but smart second-order online feature selection (SOFS) algorithm. It is not only extremely efficient and scalable to large scale and ultra-high dimensionality, but also effective to address this open challenge. Compared to existing online OFS methods, the complexity is significantly reduced to being linear to the average number of non-zero features per instance, rather than the full feature dimensionality. In particular, the proposed algorithm exploits the recent advances of second-order online learning techniques [Crammer et al. 2009b; Dredze et al. 2008]. It tries to select the most confident weights and achieves highly competitive learning accuracy even compared with state-of-the-art batch FS methods.

Our main contributions can be summarized as follows:

—We propose a new effective SOFS algorithm for large-scale ultra-high-dimensional sparse data.
—We propose fast algorithms for both existing first-order and our proposed SOFS algorithms. In particular, the computational complexity of the proposed second-order algorithm is reduced from being linear to the whole dimensionality to being linear to the average number of non-zero features.
—We conduct extensive experiments to verify the effectiveness and efficiency of the proposed algorithm.

The rest of this article is organized as follows: Section 2 reviews the related work. Section 3 presents the proposed method in detail. Section 4 shows our empirical evaluations. Section 5 draws our conclusions and discusses our future work.

2. RELATED WORK

Our work is related to FS and online learning. We review the related work in each below.

FS methods have been extensively studied in literature [Bolón-Canedo et al. 2015; Hua et al. 2009; Liu and Yu 2005; Zhao et al. 2013]. They can be roughly grouped

into three categories: *Filter*, *Wrapper*, and *Embedded* methods. *Filter* methods rely on characteristics of data such as correlation, distance, and information gain without assuming specific classifiers [Jiang et al. 2015; Jiang and Wang 2016; Li et al. 2016; Yu and Liu 2003]. Yang et al. [2015] explore the combination effectiveness of selected features with a multiple kernel learning approach, since traditional methods suffer from the "monotonic" problem. The drawback of the *Filter* methods is that they ignore the effect of selected features on the performance of the induction algorithm. By contrast, *Wrapper* methods employ a predetermined classifier to evaluate the quality of selected features [Kohavi and John 1997]. They search for a subset of features and then evaluate their classification performances repeatedly. They often yield better performance for the chosen classifier, but are computationally intensive. *Embedded* methods integrate feature selection into the model training process [Le Thi et al. 2014; Maldonado et al. 2014; Pappu et al. 2015; Xu et al. 2009]. These methods aim to make a tradeoff between the efficiency of *filter* methods and the predictive accuracy of *wrapper* methods. However, their selected features might not be suitable for other classifiers.

Many studies have attempted to address OFS in diverse ways. Some aim to handle streaming features arriving sequentially to the classifier [Glocer et al. 2005; Perkins and Theiler 2003; Wu et al. 2010]. They follow the stream learning setting and return a trained model at each time step given the observed features. However, these methods assume all the training instances must be given. This is often unrealistic for many online applications. In more general case where data arrive sequentially and continuously, Huang et al. [2015] tackle OFS from an unsupervised perspective. Our work is more closely related to another OFS setting by Wang et al. [2014]. Despite its considerable advantages in efficiency and scalability over batch FS methods, it is still slow when applied to large-scale FS tasks with ultra-high dimensionality.

Our work is also related to online learning [Hoi et al. 2014]. A variety of online algorithms have been proposed in the past few years, ranging from classical first-order algorithms [Crammer et al. 2006; Rosenblatt 1958; Zinkevich 2003] to recent second-order algorithms [Crammer et al. 2013; Ding et al. 2015; Dredze et al. 2008; Lu et al. 2013; Wang et al. 2012, 2016]. In general, these algorithms require to access and explore the full set of features. They are not directly applicable to OFS tasks for selecting a fixed number of active features. Another closely related online learning method is sparse online learning [Duchi et al. 2011; Gao et al. 2014; Wang et al. 2014]. It aims to learn a sparse linear classifier from training data in high-dimensional space. Yang et al. [2013] proposed a sparse online learning framework to solve the multi-task feature selection problem for efficiency and memory concern. Despite the extensive efforts, most of these works usually impose a soft constraint, such as ℓ_1-regularization, onto the objective function for promoting sparsity. They do not directly solve an OFS task that requires a hard constraint on the number of active dimensions in the learned classifier. In this article, we explore online learning techniques for advancing the state of the art of OFS algorithms.

3. ONLINE FEATURE SELECTION

In this section, we present a novel OFS method. We first describe the problem setting. Then, we briefly introduce the existing first-order online feature selection (FOFS) methods. Finally, we present the proposed SOFS method in detail.

3.1. Problem Setting

Without loss of generality, this article first investigates the problem of binary linear classification tasks. We will extend the solution to the multi-class setting in Section 3.6. Let $\{(\mathbf{x}^t, y^t)|t = 1, \ldots, T\}$ be a sequence of data instances received sequentially over the training process. Each $\mathbf{x}^t \in \mathbb{R}^d$ is a d-dimensional vector and $y^t \in \{+1, -1\}$. An online

learner will learn a classifier with the same dimensionality $\mathbf{w} \in \mathbb{R}^d$. At each time t, a learner receives an example $\mathbf{x}^t$. It then predicts the class label $\hat{y}^t \in \{+1, -1\}$ based on the current model, i.e., the linear weight vector $\mathbf{w}^t$:

$$\hat{y}^t = \text{sign}(\mathbf{w}^t \cdot \mathbf{x}^t). \tag{1}$$

After making the prediction, the true label y^t will be revealed. The learner can measure the loss $\ell(\mathbf{w}^t)$ suffered with respect to $(\mathbf{x}^t, y^t)$, which is the difference between the prediction and the true label. At the end of each iteration, the learner will update the weight vector $\mathbf{w}^t$ according to some updating rules. For example, the OGD algorithm updates the model as follows:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \eta^t y^t \mathbf{x}^t, \tag{2}$$

where η^t is the learning rate at time t. According to the different updating rules, online learning algorithms can be grouped into two categories:

—*First-order algorithms*, which are essentially gradient descent methods [Crammer et al. 2006].
—*Second-order algorithms*, which exploit geometrical properties of the input data [Crammer et al. 2013] or construct approximations to the Hessian of the objective functions [Duchi et al. 2011].

In the setting of OFS, we need to select a relatively small number of elements in $\mathbf{w}$ and set the others to zero. In other words, we impose the following constraint on the L_0 norm of $\mathbf{w}$:

$$\|\mathbf{w}\|_0 \leq B, \quad \|\mathbf{w}\|_0 = \sum_{i=1}^{d} {w_i}^0, \tag{3}$$

where B is the predefined constant. Consequently, at most B features of $\mathbf{x}$ will be used for prediction.

3.2. First-Order Online Feature Selection

One of the most straightforward approaches to OFS is to apply the Perceptron algorithm via truncation (PET) [Wang et al. 2014]. Specifically, at each step, the classifier first predicts the label $\hat{y}^t$ with $\mathbf{w}^t$. If $\hat{y}^t$ is correct, then $\mathbf{w}^{t+1} = \mathbf{w}^t$; otherwise, the classifier will update $\mathbf{w}^t$ by the Perceptron rule: $\hat{\mathbf{w}}^{t+1} = \mathbf{w}^t + \eta y^t \mathbf{x}^t$. It will be further truncated by keeping the largest B absolute values of $\hat{\mathbf{w}}^{t+1}$ and setting the rest to zero. The truncated classifier, denoted by $\mathbf{w}^{t+1}$, will be used to predict the next data instance. Algorithm 1 shows the framework of PET and Algorithm 2 shows the truncation step for OFS.

As analyzed in Wang et al. [2014], the above simple approach does not always work well in practice. In particular, it cannot guarantee a small number of mistakes since it fails to ensure that the numerical values of truncated elements are sufficiently small. This will lead to a non-trivial loss of accuracy. Consequently, the authors in Wang et al. [2014] proposed a FOFS algorithm by applying sparse projection before truncation. FOFS guarantees the resulting classifier $\mathbf{w}^t$ is restricted into an ℓ_1-ball at each step. Algorithm 3 shows the details of the FOFS algorithm.

3.3. Second-Order Online Feature Selection

In general, the FOFS methods have a linear time complexity with respect to the feature dimensionality. It could be very slow for ultra-high-dimensional data. Besides, in cases where the different dimensions of the input data are not in the same scale, this method may remove potentially informative features. As shown in equation (1), the prediction is not only dependent on the weight vector, but also on the input data. Even though

ALGORITHM 1: PET: Framework of Perceptron with Truncation

Input: B – number of features to select, η – learning rate
Output: weight vector $\mathbf{w}^T$
Initialize: $\mathbf{w}^1 = \mathbf{0}$;
for t *in* $\{1, \ldots, T\}$ **do**
  Receive $\mathbf{x}^t \in \mathbb{R}^d$ and predict $\hat{y}^t = \text{sign}(\mathbf{w}^t \cdot \mathbf{x}^t)$;
  Receive true label y^t;
  Suffer loss $\ell(\mathbf{w}^t)$;
  if $\ell(\mathbf{w}^t) > 0$ **then**
    $\hat{\mathbf{w}}^{t+1} = \mathbf{w}^t + \eta y^t \mathbf{x}^t$;
    $\mathbf{w}^{t+1} = \text{Truncate}(\hat{\mathbf{w}}^{t+1}, B)$;
  end
end

ALGORITHM 2: Truncate

Input: $\hat{\mathbf{w}}$ – weight vector, B – number of features to select
Output: truncated weight vector $\mathbf{w}$
if $\|\hat{\mathbf{w}}\|_0 > B$ **then**
  $\mathbf{w} = \hat{\mathbf{w}}$ with every element but the B largest absolute ones set to zero;
else
  $\mathbf{w} = \hat{\mathbf{w}}$;
end

ALGORITHM 3: FOFS: First-Order OFS via Sparse Projection

Input: B – number of features to select, η – constant learning rate, λ – regularization parameter
Output: truncated weight vector $\mathbf{w}$
$\tilde{\mathbf{w}}^{t+1} = (1 - \lambda\eta)\mathbf{w}^t + \eta y^t \mathbf{x}^t$;
$\hat{\mathbf{w}}^{t+1} = \min\{1, \frac{\frac{1}{\sqrt{\lambda}}}{\|\tilde{\mathbf{w}}^{t+1}\|_2}\}\tilde{\mathbf{w}}^{t+1}$;
$\mathbf{w}^{t+1} = \text{Truncate}(\hat{\mathbf{w}}^{t+1}, B)$;

$|w_i| < |w_j|$, it is not guaranteed that $w_i * E(x_i) < w_j * E(x_j)$, where $E(x_i)$ is the expectation of x_i. To overcome these limitations, we propose a SOFS method by exploring the recent advances of second-order online learning techniques.

The confidence-weighted (CW) method [Dredze et al. 2008] assumes that the weight vector of the linear classifier follows a Gaussian distribution $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. The confidence of weights is represented by the diagonal elements of the covariance matrix Σ. The smaller Σ_{jj}, the more confidence we have in the mean value of weight w_j. Before observing any samples, all weights are of the same confidence or uncertainty. In the CW learning process, given an observed training example $(\mathbf{x}^t, y^t)$, CW makes an update by ensuring that the probability of making correct prediction on $\mathbf{x}^t$ is bigger than a threshold τ. Meanwhile, CW tries to stay close to the previous distribution. The solution for the update can be cast into the following optimization problem:

$$
\begin{aligned}
(\hat{\boldsymbol{\mu}}^{t+1}, \Sigma^{t+1}) \;=\; & \arg\min_{\boldsymbol{\mu}, \Sigma} \mathrm{D}_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}, \Sigma), \mathcal{N}(\boldsymbol{\mu}^t, \Sigma^t)) \\
& s.t. \quad \Pr_{\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)}[y^t(\mathbf{w} \cdot \mathbf{x}^t) \geq 0] \geq \tau, \qquad (4)
\end{aligned}
$$

ALGORITHM 4: Framework of the Second-Order Online Feature Selection

Input: B – number of features to select, γ – regularization parameter
Output: weight vector $\boldsymbol{\mu}^T$ and confidence Σ^T
Initialize: $\boldsymbol{\mu}^1 = \mathbf{0}$, $\Sigma^1 = I$;
for t *in* $\{1, \ldots, T\}$ **do**
  Receive $\mathbf{x}^t \in \mathbb{R}^d$ and predict $\hat{y}^t = \text{sign}(\boldsymbol{\mu}^t \cdot \mathbf{x}^t)$;
  Receive true label y^t;
  Suffer loss $\ell(\boldsymbol{\mu}^t) = \max(0, 1 - y^t(\boldsymbol{\mu}^t \cdot \mathbf{x}^t))^2$;
  if $\ell(\boldsymbol{\mu}^t) > 0$ **then**
    Calculate β^t, $\mathbf{g}^t$ by equation (8);
    for $j = 1, \ldots, d$ **do**
      $\hat{\mu}_j^{t+1} = \mu_j^t - \frac{1}{2}\beta^t \Sigma_{jj}^t g_j^t$, $(\Sigma_{jj}^{t+1})^{-1} = (\Sigma_{jj}^t)^{-1} + \frac{(x_j^t)^2}{\gamma}$;
    end
    for $j = 1, \ldots, d$ **do**
      if Σ_{jj}^{t+1} *in smallest* B *diagonal elements* **then**
        $\mu_j^{t+1} = \hat{\mu}_j^{t+1}$;
      else
        $\mu_j^{t+1} = 0$;
      end
    end
  end
end

where $\mathrm{D}_{\mathrm{KL}}(*, *)$ is the Kullback–Leibler (KL) divergence. The KL divergence of two Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ and $\mathcal{N}(\boldsymbol{\mu}^t, \Sigma^t)$ is defined as

$$\begin{aligned}\mathrm{D}_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}, \Sigma), \mathcal{N}(\boldsymbol{\mu}^t, \Sigma^t)) \;=\; & \frac{1}{2}\log\frac{\det\Sigma^t}{\det\Sigma} + \frac{1}{2}\mathrm{Tr}((\Sigma^t)^{-1}\Sigma) \\ & + \frac{1}{2}(\boldsymbol{\mu}^t - \boldsymbol{\mu})^T(\Sigma^t)^{-1}(\boldsymbol{\mu}^t - \boldsymbol{\mu}) - \frac{d}{2}. \end{aligned} \tag{5}$$

The constraint in (4) can be rewritten as $y^t(\boldsymbol{\mu} \cdot \mathbf{x}^t) \geq \phi\sqrt{(\mathbf{x}^t)^T \Sigma \mathbf{x}^t}$, where $\phi = \Phi^{-1}(\tau)$ (Φ is the cumulative function of normal distribution). Various approaches have been proposed to solve the optimization problem in (4). In this work, we explore AROW [Crammer et al. 2013], which performs adaptive regularization of the prediction function for each new observation. It has shown to be more robust in handling label noises than the original CW algorithms:

$$(\hat{\boldsymbol{\mu}}^{t+1}, \Sigma^{t+1}) = \arg\min_{\boldsymbol{\mu}, \Sigma} \left\{ \mathrm{D}_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}, \Sigma), \mathcal{N}(\boldsymbol{\mu}^t, \Sigma^t)) + \frac{1}{2\gamma}\ell^t(\boldsymbol{\mu}) + \frac{1}{2\gamma}(\mathbf{x}^t)^T \Sigma \mathbf{x}^t \right\}, \tag{6}$$

where $\gamma > 0$ is a regularization parameter. $\ell^t(\boldsymbol{\mu})$ is the squared hinge loss function:

$$\ell^t(\boldsymbol{\mu}) = \max(0, 1 - y^t(\boldsymbol{\mu} \cdot \mathbf{x}^t))^2. \tag{7}$$

The problem in (6) can be solved with the closed-form solutions as follows:

$$\begin{aligned} \beta^t \;&=\; \tfrac{1}{(\mathbf{x}^t)^T \Sigma^t \mathbf{x}^t + \gamma} \quad \mathbf{g}^t = -2\max\left(0, 1 - y^t(\boldsymbol{\mu}^t \cdot \mathbf{x}^t) y^t \mathbf{x}^t \right. \\ \hat{\boldsymbol{\mu}}^{t+1} \;&=\; \boldsymbol{\mu}^t - \tfrac{1}{2}\beta^t \Sigma^t \mathbf{g}^t \quad (\Sigma^{t+1})^{-1} = (\Sigma^t)^{-1} + \tfrac{\mathrm{diag}(\mathbf{x}^t(\mathbf{x}^t)^T)}{\gamma}. \end{aligned} \tag{8}$$

Note that only the diagonal elements of the covariance matrix Σ are considered. From the efficiency perspective, maintaining a full covariance matrix requires $O(d^2)$ memory space and $O(d^2)$ computational complexity. It is impractical to handle large-scale ultra-high-dimensional data in such case. From the learning ability perspective,

it is explored that given enough data, the diagonal approximation is able to outperform the full version. This is because the same ability to adapt to data co-dependencies that helps full CW learning during the early rounds leads to adapt to noise as it approaches the optimal weight vector when the data are not separable [Ma et al. 2010].

Unlike the FOFS methods that select the important features based on the magnitudes of the weight vector, the key idea of the proposed SOFS technique is to keep the B most confidence features by exploiting the second-order information of the classifier. Specifically, in the online learning process, when the loss for a training instance $(\mathbf{x}^t, y^t)$ is non-zero, we only update the most confident B weight variables whose covariance values Σ_{jj} are among the B smallest. All the other weights are set to zero. Algorithm 4 shows the details of the proposed SOFS algorithm.

3.4. Efficient Algorithms

A common drawback with many existing OFS methods is the high computational cost over ultra-high dimensions. Specifically, one of the major time-consuming procedures is to select the largest or smallest B elements from a d-dimensional array (the absolute weight vector in FOFS and the diagonal vector of Σ in SOFS). Instead of sorting all the elements at each step as in the previous study [Wang et al. 2014], we propose an efficient and scalable solution by employing the MinHeap-based implementation for PET and FOFS. Besides, with a similar MaxHeap-based implementation, we further reduce the complexity by exploiting the *monotonic decreasing* property of SOFS as shown in Section 3.4.2.

3.4.1. Efficient Algorithms for First-Order OFS. To find out the B largest values from a d-dimensional array (either PET in Algorithm 1 or FOFS in Algorithm 3), a straightforward solution is to sort the d values and select the top B elements. To improve the efficiency, we build a MinHeap to store the B largest absolute elements of the weight vector $\mathbf{w}^t$. At each learning step, whenever the classifier is updated, we make the following two-step updates to figure out the B largest elements:

—Adjust positions of the elements that already exist in the heap to maintain the heap.
—Compare each element that is not in the heap with the heap limit. If the value is smaller than the heap limit, set the value of the element to zero. Otherwise, replace the root node of the heap by the element. After that, adjust the position of the root node with its child nodes recursively to maintain the heap property. The value of the original root node is set to zero.

Algorithm 5 shows the detailed procedures of the improved algorithms for the FOFS algorithm. Fast algorithm for PET is similar. We note that the improved algorithms have never been proposed in previous studies.

To illustrate why the above steps work, we need to prove that the largest B elements will be in the MinHeap after each iteration. Let $h_1, \ldots, h_B$ denote the indices of elements that are already in the heap. The rest elements are with indices $h_{B+1}, \ldots, h_d$. In the first step, $w_{h_1}, \ldots, w_{h_B}$ are re-organized to maintain the MinHeap. We have the following two propositions.

PROPOSITION 3.1. *If $w_{h_i}, \forall i \in [1, B]$ is still among the largest B elements after updating the model, w_{h_i} will not be replaced out of the heap.*

PROPOSITION 3.2. *If $w_{h_i}, \forall i \in (B, d]$ is among the largest B elements after updating the model, w_{h_i} will be replaced into the heap.*

PROOF. To prove the proposition 3.1, if w_{h_i} is not the smallest one among the largest B elements, w_{h_i} will never become the root node due to the heap property. Thus, it will

ALGORITHM 5: Fast Algorithm for First-Order OFS

Input: B – number of features to select, η – learning rate, λ – regularization parameter .
Output: weight vector $\mathbf{w}^T$
Initialize: $\mathbf{w}^1 = \mathbf{0}$, $\mathbf{v}^1 = (|w_1^1|, \ldots, |w_d^1|) = \mathbf{0}$, MinHeap H on $\mathbf{v}^1$ with size B;
for t *in* $\{1, \ldots, T\}$ **do**
  Receive $\mathbf{x}^t \in \mathbb{R}^d$ and predict $\hat{y}^t = \text{sign}(\mathbf{w}^t \cdot \mathbf{x}^t)$;
  Receive true label y^t;
  Suffer loss $\ell(\mathbf{w}^t)$;
  if $\ell(\mathbf{w}^t) > 0$ **then**
    $\tilde{\mathbf{w}}^{t+1} = (1 - \lambda\eta)\mathbf{w}^t + \eta y^t \mathbf{x}^t$;
    $\mathbf{w}^{t+1} = \min\{1, \frac{\frac{1}{\sqrt{\lambda}}}{\|\tilde{\mathbf{w}}^{t+1}\|_2}\}\tilde{\mathbf{w}}^{t+1}$;
    $\mathbf{v}^{t+1} = (|w_1^{t+1}|, \ldots, |w_d^{t+1}|)$;
    adjust H to maintain the MinHeap;
    for $j = 1, \ldots, d$, $v_j^{t+1} \notin H$ **do**
      if $v_j^{t+1} > H_{min}$ **then**
        get the index of H_{min}, denoted by s;
        $w_s^{t+1} = 0$;
        replace H_{min} by v_j^{t+1};
        adjust the root node with its childnodes recursively to maintain the MinHeap;
      else
        $w_j^{t+1} = 0$;
      end
    end
  end
end

never be replaced out of the heap. Otherwise, w_{h_i} is the smallest and the rest $B-1$ are no smaller than w_{h_i}. Combined with the assumption that w_{h_i} is still among the largest B elements, the rest $d-B$ elements outside of the heap should all be no bigger than w_{h_i}. As a result, w_{h_i} will not be replaced out of the heap in both cases. As to proposition 3.2, we can easily conclude that the root node of the heap is smaller than w_{h_i} since w_{h_i} is among the largest B elements. As a result, w_{h_i} will always be replaced into the heap as long as the heap is well maintained. □

3.4.2. Efficient Algorithms for Second-Order OFS. Despite the better implementation, computational complexity of the previous algorithms is still linear w.r.t. the feature dimensionality d. In this section, we present an efficient algorithm for SOFS, whose computational complexity depends linearly on the number of non-zero features of each example m, instead of the feature dimensionality d. This makes it extremely efficient and scalable when handling real-world high-dimensional sparse data. Before presenting the proposed algorithm, we first introduce the following proposition for the *monotonic decreasing* property of Σ^t, a property that is critical to the proposed algorithm.

PROPOSITION 3.3 (MONOTONIC DECREASING). *Given* Σ^t *computed by (8),* $\forall t$ *and* $\forall j \in [1, d]$, $\Sigma_{jj}^{t+1} \leq \Sigma_{jj}^t$.

It is not difficult to verify the above proposition by noticing $\text{diag}(\mathbf{x}^t(\mathbf{x}^t)^T)/\gamma$ is always non-negative. Using this important property, we can develop a fast algorithm for the SOFS method.

ALGORITHM 6: SOFS: Fast Algorithm for Second-Order OFS

Input: B – number of features to select, γ – regularization parameter
Output: Output: weight vector $\boldsymbol{\mu}^T$ and confidence Σ^T
Initialize: $\boldsymbol{\mu}^1 = \mathbf{0}$, $\Sigma^1 = I$, MaxHeap H on Σ^1 with size B;
for t *in* $\{1, \ldots, T\}$ **do**
  Receive $\mathbf{x}^t \in \mathbb{R}^n$ and predict $\hat{y}^t = \text{sign}(\boldsymbol{\mu}^t \cdot \mathbf{x}^t)$;
  Receive true label y^t;
  Suffer loss $\ell(\boldsymbol{\mu}^t) = \max(0, 1 - y^t(\boldsymbol{\mu}^t \cdot \mathbf{x}^t))^2$;
  if $\ell(\boldsymbol{\mu}^t) > 0$ **then**
    Calculate β^t, $\mathbf{g}^t$ by (8);
    for $j = 1, \ldots, d, \mathbf{x}_{t,j} \neq 0$ **do**
      $\mu_j^{t+1} = \mu_j^t - \frac{1}{2}\beta^t \Sigma_{jj}^t g_j^t, \quad (\Sigma_{jj}^{t+1})^{-1} = (\Sigma_{jj}^t)^{-1} + \frac{(x_j^t)^2}{\gamma}$;
      if Σ_{jj}^{t+1} *in* H **then**
        adjust Σ_{jj}^{t+1} recursively with its child nodes to maintain the MaxHeap;
      end
    end
    for $j = 1, \ldots, d, x_j^t \neq 0, \Sigma_{jj}^{t+1} \notin H$ **do**
      if $\Sigma_{jj}^{t+1} < H_{max}$ **then**
        get the index of H_{max}, denoted by s;
        $\mu_s^{t+1} = 0$;
        replace H_{max} by Σ_{jj}^{t+1};
        adjust the root node recursively with its child nodes to maintain the MaxHeap;
      else
        $\mu_j^{t+1} = 0$;
      end
    end
  end
end

Similar to the previous solution, we build a MaxHeap data structure to store the B smallest diagonal values of covariance Σ^t. The monotonic decreasing property of Σ^t leads to two major benefits in saving computational cost:

—When maintaining the heap in step 1, we do not need to update the positions of all the B elements. Whenever an element is changed, we only need to compare and update it with its child nodes since we know its parent node is guaranteed to be bigger than the current element.
—In step 2, we do not need to check those unchanged elements outside the heap to see if they are among the B smallest elements. This is because the root node of the MaxHeap is non-increasing. The unchanged elements should still be bigger than the root node after updating the model.

Algorithm 6 shows the details of the proposed fast algorithm for SOFS.

3.5. Analysis of Time and Space Complexity

The above proposed technique significantly improves the efficiency of OFS algorithms. We now analyze the computational complexity of the above algorithms.

Let us denote the dimensionality of the weight vector by d, and m the average number of non-zero features of each sample. In the worst case, each updating step of PET requires:

—$2m$ for calculating the loss and updating the model;
—m for calculating the absolute value of the model;
—$B \log B$ for maintaining the MinHeap;
—$(d - B) \log B$ for finding out the largest B elements and maintaining the MinHeap;
—$d - B$ for setting the corresponding elements to zero.

The overall computational complexity of PET at each step is $\{3m + d - B + d \log B\}$.

FOFS is similar to PET, which requires:

—$2m$ for calculating the loss and updating the model;
—d for calculating the norm;
—d for sparse projection;
—d for calculating the absolute values of the model;
—$B \log B$ for maintaining the MinHeap;
—$(d - B) \log B$ for finding out the largest B elements and maintaining the MinHeap;
—$d - B$ for setting the corresponding elements to zero.

The overall complexity of FOFS at each step is $\{2m + 4d - B + d \log B\}$, which is much more computationally expensive than PET for high dimensional data.

The updating of the proposed SOFS requires:

—$3m$ for calculating the loss, updating the model and the covariance;
—$m \log B$ for maintaining the MaxHeap since only m values changed;
—m for setting the corresponding elements to zero.

The computational complexity of SOFS at each step is reduced to $\{4m + m \log B\}$, making it far more efficient and scalable when handling ultra-high-dimensional sparse data where $m \ll d$ and $B \ll d$. In the worst case where $m \approx d$, the cost of the maintaining step of SOFS is approximate to PET, but still much smaller than FOFS.

For space complexity, we only consider the space required by the classifiers. Storage for data loading is omitted here. In our implementation, the input data are stored in sparse arrays of index-value format, while the model parameters are represented by dense vectors for efficiency concern. Both PET and FOFS require to keep the weight vector $\mathbf{w}$ and its absolute vector in memory. The space complexity is $O(2d)$. SOFS also has the space complexity of $O(2d)$ for keeping the weight vector and the diagonal confidence matrix Σ in memory. As a result, SOFS has the same space complexity as the FOFS algorithms.

3.6. Second-Order Multi-Class Online Feature Selection

In the multi-class setting, each training example is associated with a label $y \in \{0, 1, \ldots, k-1\}$ for k classes. We adopt the one-vs.-the-rest strategy to extend the SOFS to the multi-class setting. As suggested in [Crammer et al. 2009a], the distribution of the confidence weighted model is similar to the binary case, $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma), \boldsymbol{\mu} \in \mathbb{R}^{kd}, \Sigma \in \mathbb{R}^{kd \times kd}$. We introduce a new label-dependent feature,

$$\psi(\mathbf{x}, i) = [\mathbf{0}^T, \ldots, \mathbf{x}^T, \ldots, \mathbf{0}^T]^T,$$

where only the ith position $\psi(\mathbf{x}, i)$ is $\mathbf{x}$ and others are $\mathbf{0}$ ($\mathbf{0}, \mathbf{x} \in \mathbb{R}^d$). At each step, the classifier receives a new example $\mathbf{x}^t$ and predicts the label $\hat{y}^t = \arg\max_{i=0}^{k-1} \boldsymbol{\mu}^t \cdot \psi(\mathbf{x}, i)$. The classifier suffers a squared hinge loss:

$$\ell(\boldsymbol{\mu}^t) = \max(0, 1 - \boldsymbol{\mu}^t \cdot \Delta\psi^t)^2, \tag{9}$$

where $\Delta\psi^t$ depends on the multi-class updating strategy.

For max-score multi-class update,

$$\Delta\psi^t = \psi(\mathbf{x}^t, y^t) - \psi(\mathbf{x}^t, \arg\max_{i=0, i\neq y^t}^{k-1} \boldsymbol{\mu}^t \cdot \psi(\mathbf{x}, i)). \tag{10}$$

For uniform multi-class update, let

$$E^t = \{i \neq y^t : \boldsymbol{\mu}^t \cdot \psi(\mathbf{x}^t, i) \geq \boldsymbol{\mu}^t \cdot \psi(\mathbf{x}^t, y^t)\},$$

we have

$$\Delta\psi^t = \sum_{i=0}^{k-1} \alpha_i^t \psi(\mathbf{x}^t, i), \quad \alpha_i^t = \begin{cases} -1/|E^t| & i \in E^t \\ 1 & \text{if} \quad i = y^t \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

The update is performed as follows:

$$(\hat{\boldsymbol{\mu}}^{t+1}, \Sigma^{t+1}) = \arg\min_{\boldsymbol{\mu}, \Sigma} \left\{ \mathrm{D}_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{\mu}, \Sigma), \mathcal{N}(\boldsymbol{\mu}^t, \Sigma^t)) + \frac{1}{2\gamma}\ell(\boldsymbol{\mu}) + \frac{1}{2\gamma}(\Delta\psi^t)^T \Sigma \Delta\psi^t \right\}. \tag{12}$$

The closed-form solution is similar to equation (8), except that $y^t\mathbf{x}^t$ is replaced by $\Delta\psi^t$.

To select features, we keep the B features that are most confident. In the multi-class setting, confidence of a feature depends on the k binary classifiers in the one-vs-the-rest strategy. The confidence of the jth feature is measured by $C_j = k - \sum_{i=0}^{k-1} \Sigma_{ij,ij}$. We update the weight vector only for those whose confidence C_j is among the B largest. All the other weights are set to zero. The details of this algorithm are similar to those of Algorithm 6, with some replacement of $y^t\mathbf{x}^t$ by $\Delta\psi^t$. The time complexity of SOFS in the multi-class setting is k times that of the binary case.

Note that the sum $\sum_{i=0}^{k-1} \Sigma_{ij,ij}$ is also monotonic decreasing in the multi-class setting, as shown in the following proposition, which can be easily verified.

PROPOSITION 3.4 (MONOTONIC DECREASING). *Given* Σ^t *computed by* (12), $\forall t$ *and* $\forall j \in [1, d]$, $\sum_{i=0}^{k-1} \Sigma_{ij,ij}^t \leq \sum_{i=0}^{k-1} \Sigma_{ij,ij}^{t+1}$.

As a result, the fast algorithm in the binary classification also works in the multi-class setting.

4. EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the performance of the proposed SOFS algorithm. We compare with existing FS algorithms on both synthetic and real data, from medium scale to large scale.

4.1. Experimental Setup

For the family of OFS algorithms, we only run each algorithm by a single pass through the training data if without explicit indication. We compare the proposed algorithm with a set of state-of-the-art algorithms, including both online and batch FS as follows:

—PET: the baseline of OFS by Perceptron with truncation [Wang et al. 2014];
—FOFS: the state-of-the-art first-order OFS via sparse projection [Wang et al. 2014];
—mRMR: minimum Redundancy Maximum Relevance Feature Selection [Peng et al. 2005], a state-of-the-art batch FS method and its parallelized version with Graphic Processing Unit (GPU-mRMR) [Ramírez-Gallego et al. 2017].
—LIBLINEAR: a famous library for large linear classification [Fan et al. 2008]. We adopt $l1$-SVM for the *Embedded* FS in our experiments.
—FGM: a batch *Embedded* feature-generating method [Tan et al. 2014].

Table I. Summary of Synthetic data ("K,""M," and "B" are Short for Thousand, Million, and Billion, Respectively)

Dataset	#Train	#Test	Dim	IDim[a]	NDim[b]	#Feature
$\mathbf{X}_1$	100K	10K	10K	100	200	30M
$\mathbf{X}_2$	100K	10K	20K	200	400	60M
$\mathbf{X}_3$	1M	100K	1B	500	500	1B

[a]dimension of informative features per instance.
[b]dimension of noise features per instance.

For online algorithms, we use hinge loss as the loss function. A five-fold cross validation is conducted to identify the optimal parameters. The experiments are conducted 10 times with a random permutation of the dataset. For $l1$-SVM in LIBLINEAR, we tune parameter C to select different number of features. For FGM, we follow the settings in [Tan et al. 2014] and set $C = 10$ for simplicity. For mRMR, we first select a specific number of features and then use the online gradient descent (OGD) algorithm to train a classifier. We exploit the advantage of online learning that processes data sequentially. We implement the online algorithms with two parallel threads, one for data loading and the other for training. All experiments are conducted on a single server with 64GB RAM.[1] The server also features a Nvidia Titan X GPU card for the parallelized algorithm.

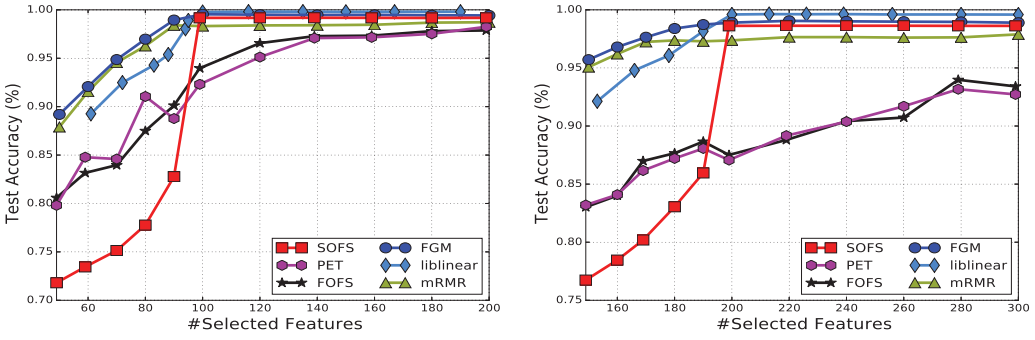
4.2. Experiments on Synthetic Data

Synthetic data. We follow the settings of FGM and generate three types of synthetic data, namely $\mathbf{X}_1 \in \mathbb{R}^{100K\times 10K}, \mathbf{X}_2 \in \mathbb{R}^{100K\times 20K}$, and $\mathbf{X}_3 \in \mathbb{R}^{1M\times 1B}$ to test the efficacy, efficiency, and scalability of the proposed algorithm. All datasets are for binary classification. Each entry is sampled from the *i.i.d.* Gaussian distribution $\mathcal{N}(0, 1)$. To simulate real data, each sample is a sparse vector. The numbers of informative features for the three datasets are 100, 200, and 500, respectively. For each sample, we randomly select 200 dimensions for $\mathbf{X}_1$, 400 dimensions for $\mathbf{X}_2$, and 500 dimensions for $\mathbf{X}_3$ as noise. To generate labels, we sample a weight vector $\mathbf{w}^*$ from the Uniform distribution $\mathcal{U}(0, 1)$ as the groundtruth. The label of each sample is determined by $y = \text{sign}(\mathbf{w}^* \cdot \mathbf{x}^*)$, where $\mathbf{x}^*$ is a sample without noise. Details of the synthetic datasets are shown in Table I.

4.2.1. Experiment on Medium-Scale Synthetic Data. We evaluate all the algorithms on $\mathbf{X}_1$ and $\mathbf{X}_2$. The largest-scale dataset $\mathbf{X}_3$ will be used to test scalability of the algorithms in the subsequent experiment. Figures 1 and 2 show the comparison of accuracy and time cost, respectively. In the following we analyze each of them in detail.

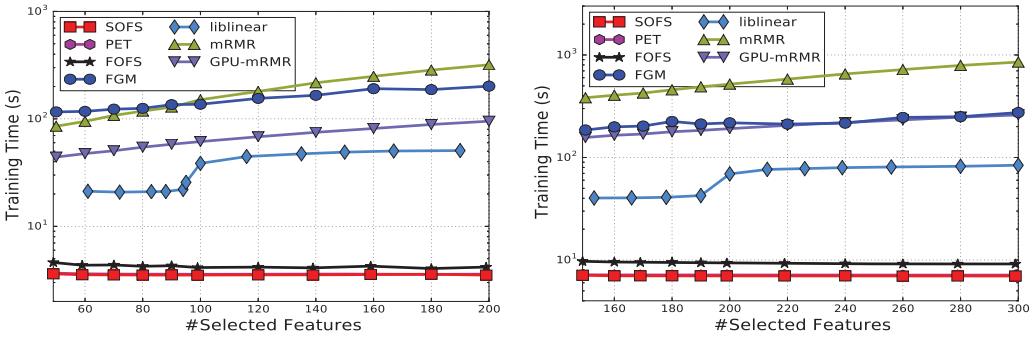
Accuracy. Figure 1 shows the comparison of accuracy. Several observations can be drawn from the figures. First, when plenty of informative features are selected(100 in $\mathbf{X}_1$ and 200 in $\mathbf{X}_2$), SOFS is comparable to batch FS algorithms. What is more, LIBLINEAR and FGM perform better than SOFS with very limited advantage. Second, when less features are selected, batching learning algorithms are superior to online algorithms. We find that FGM and mRMR work especially better than online algorithms in such case. The accuracy of SOFS is not good with insufficient features. However, it reaches the best and then saturates quickly with more features selected. Third, the two FOFS algorithms perform the worst, especially on $\mathbf{X}_2$. PET and FOFS work better than SOFS with a very few features. However, their performances cannot reach a comparable level with batch algorithms even with adequate features. To conclude, the proposed algorithm is able to identify the number of informative features of the data. Besides, it can reach a comparable performance with batch algorithm with enough features.

[1]We run the experiments on a server because batch algorithms require large RAM. The source codes is available at LIBOL online repositories: https://github.com/LIBOL/SOL/tree/master/ofs.



(a) dataset X_1 (b) dataset X_2

Fig. 1. Test accuracy versus number of selected features on synthetic datasets $\mathbf{X}_1$ and $\mathbf{X}_2$.



(a) dataset X_1 (b) dataset X_2

Fig. 2. Time cost versus number of selected features on synthetic datasets $\mathbf{X}_1$ and $\mathbf{X}_2$.

Time cost. In addition to accuracy, training efficiency is a critical issue for real-world problems. We show time cost of the algorithms in Figure 2. Generally, we can see that the batch learning scheme, though effective, is much more time-consuming than online learning algorithms. The proposed SOFS can achieve comparable test accuracy to batch algorithms in seconds. In contrast, LIBLINEAR requires about 10 times of training time. FGM and mRMR even require 100 times of training time on the $\mathbf{X}_2$ dataset. The parallelized mRMR reduces more than half of the training time compared with the non-parallel version. Among OFS, our method still requires the least training time. We find that the time cost of OFS algorithms are similar to each other on these two datasets. We will further explore their difference on larger scale and higher dimensional data to verify the analysis in Section 3.5. Nevertheless, the accuracy and time cost comparison verify that SOFS is an efficient and effective OFS algorithm.

4.2.2. Experiment on Large-Scale Synthetic Data. In this setting, we mainly test whether SOFS is scalable to ultra-high-dimensional data with billion scale features on dataset $\mathbf{X}_3$. Due to the large scale and ultra-high dimension of $\mathbf{X}_3$, we find that it may take hours or even days for existing FS algorithms to run. As a result, we force $B = 100, 200, 500$ on $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, respectively, to test whether SOFS can handle the increasing scale and dimension. Besides, we compare with two baseline online learning algorithms with full feature sets to verify the efficacy of SOFS. The two algorithms are OGD and Adaptive Regularization of Weights (AROW). The results are shown in Table II.

Table II. Scalability evaluation of SOFS

	Algorithm	$\mathbf{X}_1$	$\mathbf{X}_2$	$\mathbf{X}_3$
Time cost	OGD(s)	3.58	7.06	114.82
	AROW(s)	3.59	7.02	130.72
	SOFS(s)	**3.51**	**7.00**	**132.94**
Accuracy	OGD(%)	98.44	97.83	99.39
	AROW(%)	98.48	98.52	99.55
	SOFS(%)	**99.17**	**98.62**	**99.56**
Model sparsity	OGD(%)	0.00	0.00	83.16
	AROW(%)	0.00	0.00	72.22
	SOFS(%)	**99.00**	**99.00**	**99.99**

Texts in bold highlight the least time cost, best accuracy or highest sparsity.

Table III. Medium-Scale Real Datasets in Experiments

Dataset	Feat dim	Train no.	Test no.	Feat no.
relathe	4,322	1,000	427	87,352
pcmac	7,510	1,000	946	55,470
basehock	4,862	1,500	493	101,974
ccat	47,236	13,149	10,000	994,133
aut	20,072	40,000	22,581	1,969,407
real-sim	20,958	50,000	22,309	2,560,340

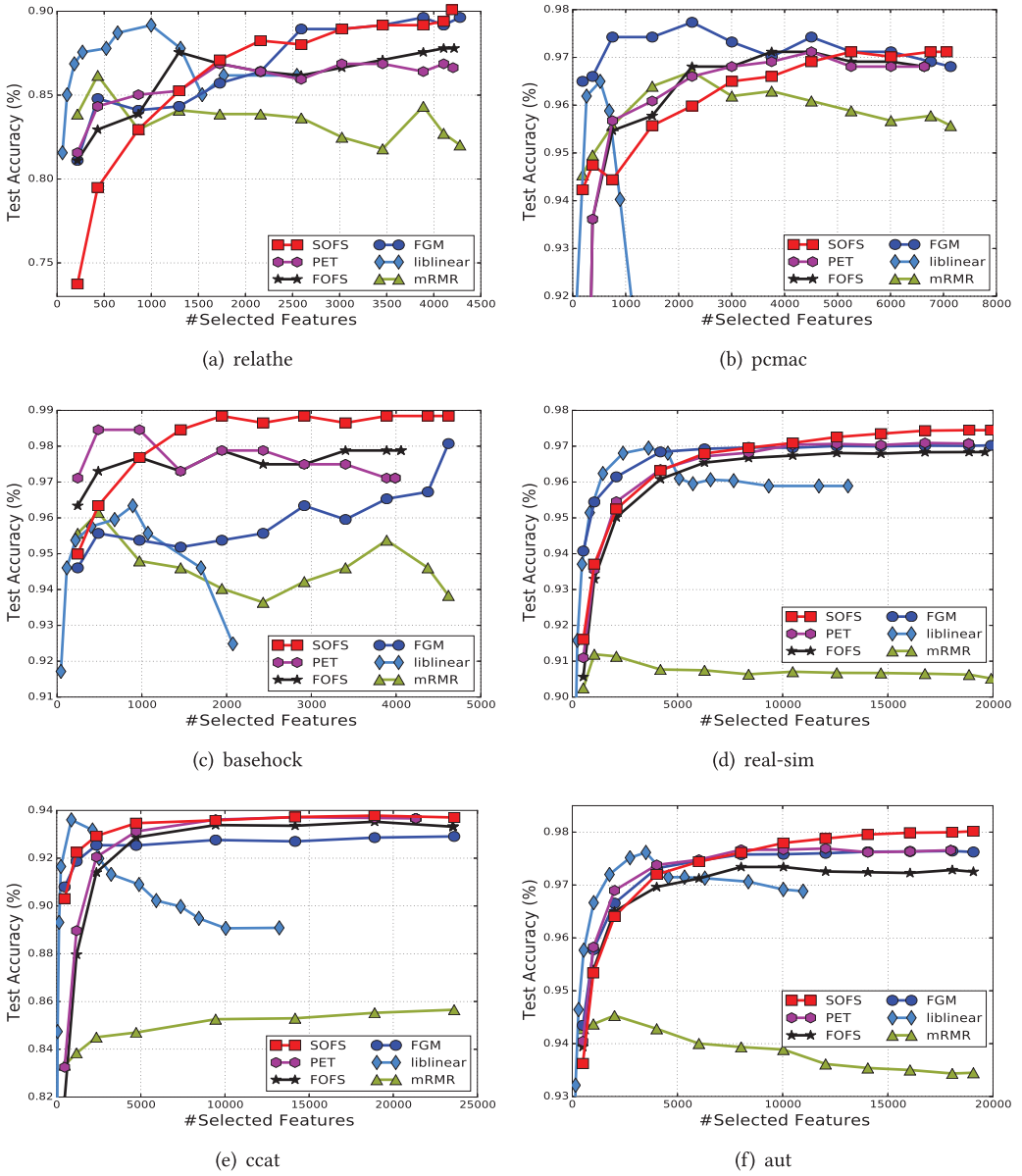
As we can see from the table, test accuracy is improved against the baseline algorithms, which verifies that removing irrelevant or noisy features can improve the prediction performance. What is more, SOFS requires less than 1% features to achieve such accuracy. The benefits are three-fold: (1) In cases where input data are dense, such a sparse classifier will reduce the prediction time cost significantly. (2) It can reduce the memory cost significantly in prediction. (3) It can significantly reduce the feature extraction time. In this example, OGD and AROW require about 1GB to represent the classifier of $\mathbf{X}_3$ (with a 4 byte float to represent each weight), while SOFS requires only 2KB. In scenarios like embedded systems where memory is limited, such a compact classifier is much more applicable and economic.

We observe that with more data samples and higher feature dimension, time cost of SOFS increases in a controllable manner. It costs only a little more than 2 minutes to train a classification model on the billion scale dataset. On the contrary, other FS algorithms suffer from the problem of *the curse of dimensionality*. For example, PET takes at least 10 hours to select 500 features from $\mathbf{X}_3$, let alone other more complicated algorithms. Besides, we would like to highlight the time cost of SOFS with the baseline algorithms. It can be found that SOFS does not incur significant extra time cost. This is due to the fact that learning and data loading run in parallel in our implementation. Since all the three algorithms are very efficient, data loading accounts for the major part of time consumption. To conclude, the efficient computation and high test accuracy indicate that our algorithm is effective and efficient in exploiting informative features on large scale ultra-high-dimensional data.

4.3. Experiments on Medium-Scale Real Datasets

In this section, we evaluate the performance of OFS algorithms on a number of medium-scale public datasets, as shown in Table III. The datasets can be downloaded either from Feature Selection website of Arizona State University[2] or SVMLin[3] (for sparse datasets).

[2]http://featureselection.asu.edu/datasets.php.
[3]http://vikas.sindhwani.org/svmlin.html.



(a) relathe (b) pcmac

(c) basehock (d) real-sim

(e) ccat (f) aut

Fig. 3. Test accuracy of feature selection algorithms on medium-scale real world data.

4.3.1. Evaluation of Accuracy. Figure 3 shows the test accuracy of different algorithms. By examining the online algorithms, we find that SOFS performs better than PET and FOFS except when the number of selected features is limited on some datasets. This observation is similar to what we found on synthetic datasets. When considering batch algorithms, SOFS is comparable to or even better than the state-of-the-art FGM algorithm with enough features. FGM performs well with a rather few features. LIBLINEAR shows a very interesting phenomenon. The test accuracy first increases

rapidly with more selected features. After a certain stage where the accuracy of other algorithms begins to saturate, the accuracy of LIBLINEAR tends to drop considerably. This is possibly because $l1$-SVM suffers the overfitting problem with small $l1$ penalty. As to mRMR, we can observe that mRMR is better when the number of features is rather small. The accuracy of SOFS increases quickly and surpasses mRMR with more selected features. However, the overall performance of mRMR is much worse than other FS algorithms.

Generally, we can find that batch learning algorithms work better with very small number of selected features. However, as more features are selected, the performance of the proposed SOFS algorithm increases quickly and achieves comparable or even better test accuracy.

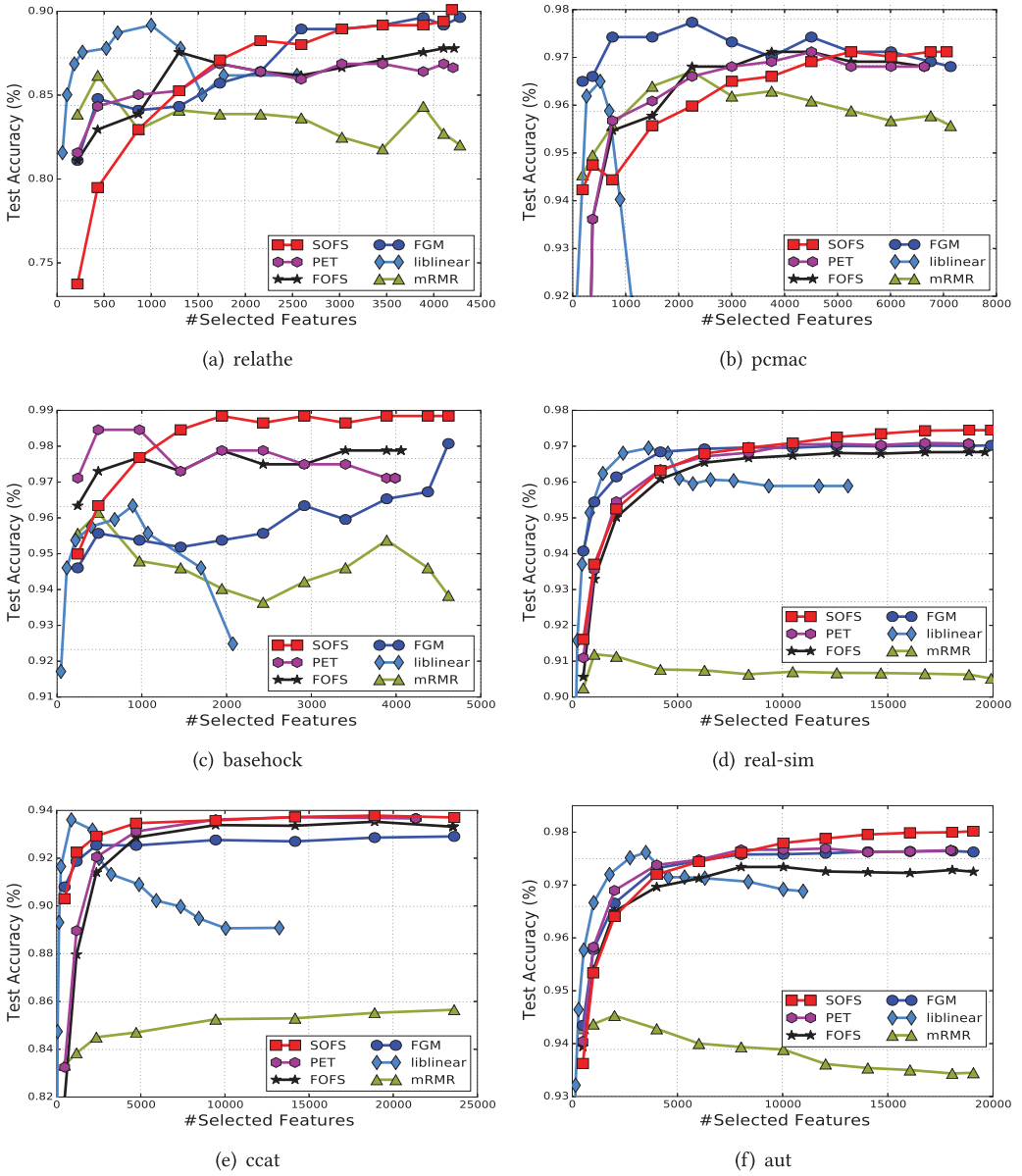
4.3.2. Evaluation of Time Cost. Figure 4 shows the time cost comparison of feature selection methods on the medium-scale data. First of all, the proposed SOFS takes the least time to do feature selection. Note the time costs on the later three datasets, which are of relatively high dimension. It shows the great advantage of our proposed algorithm on high-dimensional data. Second, there exists up to 10 times advantage over PET and FOFS. Besides, FOFS takes more time than PET, especially on the later three large datasets. Third, among the batch algorithms, LIBLINEAR is the most efficient, which is still tens of times longer than SOFS. The time cost of FGM is about an order of magnitude higher than LIBLINEAR. The most time consuming algorithm is the mRMR algorithm. On the "aut" dataset, even the parallelized algorithm requires more than 6,000 seconds to select 10,000 features, let alone the non-parallelized algorithm. Besides, we find the parallelized algorithm only accelerates the algorithm when the number of training data exceeds the data dimension ("real-sim" and "aut").

We can easily conclude that the time cost of the FS algorithms is consistent to the complexity analysis in Section 3.5. The complexity of SOFS is linearly dependent on the number of non-zero features. Though linearly dependent on the feature dimension, the complexity of PET is still more efficient than FOFS.

4.4. Experiment on Object Recognition

OFS can also be applied to the computer vision field like image and video processing [Li et al. 2017; Wang et al. 2009]. In this experiment, we apply the proposed multi-class SOFS algorithm to the real-world object recognition task. We use the VOC2007 dataset, which contains 20 object classes in realistic scenes [Everingham et al. 2007]. We first crop the objects from the images. Half of the cropped images are randomly selected as training images. The left are used as test images. To extract features, we adopt the widely used deep convolutional neural network – the VGG16 pretrained model [Simonyan and Zisserman 2014]. Each image is represented by the features of last two fully connected layers. By doing so, we obtain $12,315$ samples for training and $12,325$ samples for testing. Each sample is a $8,192$-dimensional sparse vector. The sparsity is due to the rectified linear unit activation function [Krizhevsky et al. 2012]. We omit FGM in this experiment as it is designed for binary classification only. We also omit LIBLINEAR since it can only induce sparsity on the weights rather than features.

Figure 5 shows the classification accuracy and the time cost evaluation of SOFS compared with other online and batch algorithms. We can observe that the proposed SOFS achieves the best performance among all the algorithms. The test accuracy increases rapidly with more features selected, which verifies the effectiveness of SOFS for computer vision tasks. mRMR performs the worst in this case. Another finding is that SOFS achieves the best accuracy with about 4,000 features, which is about half of the full dimension. Given that the features are generated from the fully connected layers of the deep convolutional neural network, the removed features indicate that we can



(a) relathe (b) pcmac

(c) basehock (d) real-sim

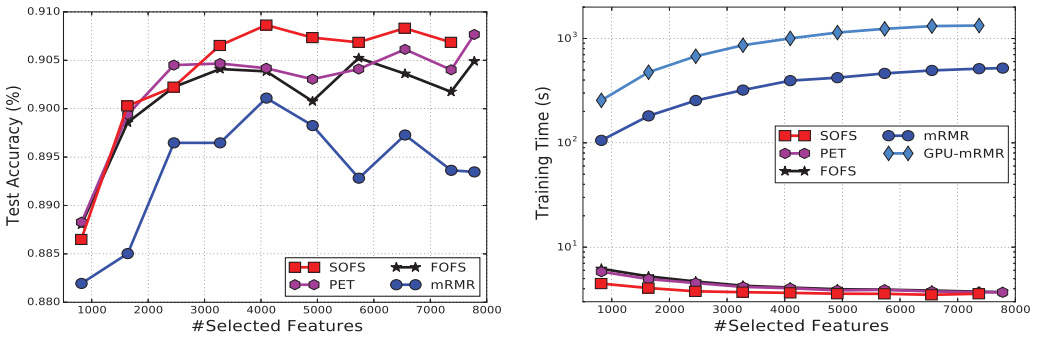
(e) ccat (f) aut

Fig. 4. Time cost of feature selection algorithms.

remove some fully connected parameters that generate the features. This will accelerate the training of deep learning and reduce the size of the deep models. Figure 5(b) again verifies the efficiency of OFS algorithms. Since the data scale and dimensionality are not high in this problem, the benefits of SOFS over PET and FOFS are not significant.

4.5. Experiments on Large-Scale Real-World Datasets

In this section, we evaluate the performance of the proposed SOFS algorithm for three large-scale text classification tasks, as shown in Table IV.



(a) Test Accuray (b) Training Time

Fig. 5. Test accuracy and time cost versus number of selected features on the caltech dataset.

Table IV. Summary of Large-Scale Real-World Datasets in Our Experiments

DataSet	Feat dim	Train no.	Test no.	Feat no.
news	1,355,191	10,000	9,996	5,513,533
rcv1	47,152	781,265	23,149	59,155,144
url	3,231,961	2,000,000	396,130	2,31,249,028

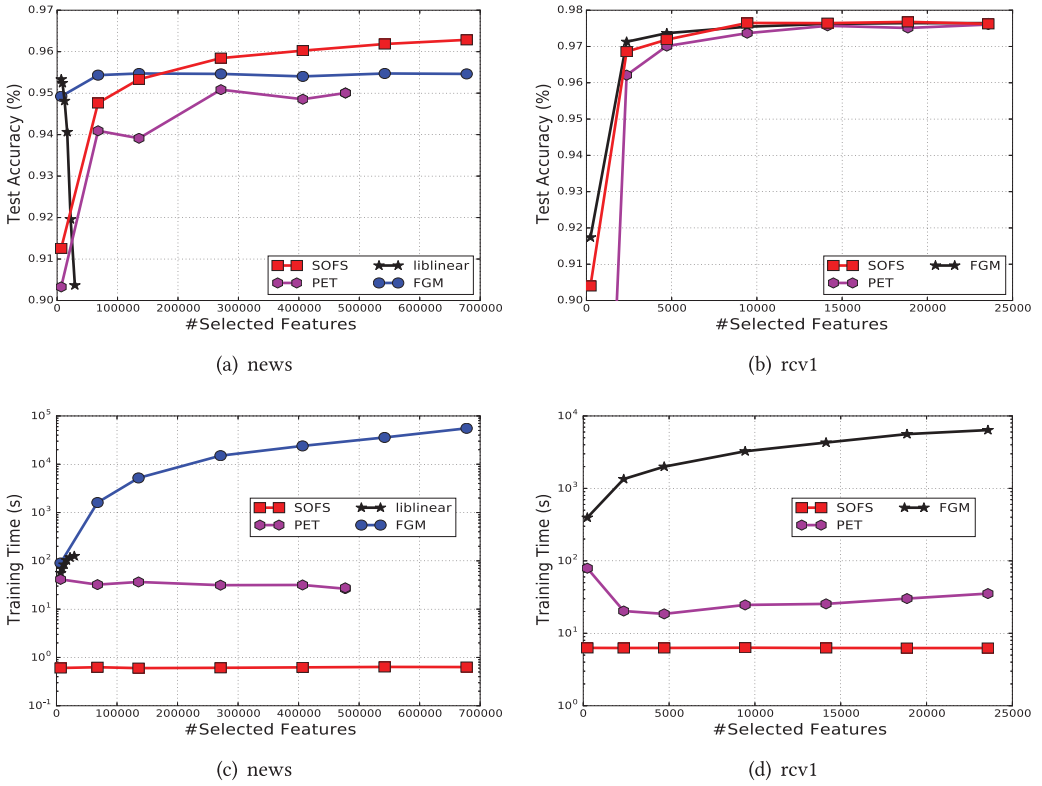
Table V. Evaluation on Large-Scale High-Dimensional Datasets (ρ is the Fraction of Selected Features)

Dataset	ρ	0.005	0.05	0.1	0.2
news	PET	90.33%(41.34s)	94.09%(32.18s)	93.91%(36.54s)	95.08%(31.37s)
	SOFS	91.26%(**0.61s**)	94.76%(**0.63s**)	95.33%(**0.60s**)	**95.84**%(**0.61s**)
	FGM	**94.92**%(90.10s)	**95.43**%(1610.53s)	**95.47**%(5206.20s)	95.46%(15055.28s)
rcv1	PET	73.18%(79.13s)	96.21%(20.30s)	97.01%(18.53s)	97.37%(24.63s)
	SOFS	90.40%(**6.29s**)	96.86%(**6.27s**)	97.19%(**6.28s**)	**97.65**%(**6.32s**)
	FGM	**91.74**%(394.98s)	**97.13**%(1346.03s)	**97.37**%(1994.78s)	97.54%(3253.97s)
url	PET	98.15%(1100.28s)	98.38%(1664.15s)	98.21%(1528.01s)	98.21%(1573.35s)
	SOFS	**98.32**%(**6.95s**)	**98.74**%(**7.05s**)	**98.92**%(**6.94s**)	**99.18**%(**6.94s**)

Texts in bold highlight the least time cost, best accuracy or highest sparsity.

The first dataset "news" (for news group classification) is high dimensional, the second "rcv1" (for text categorization) is relatively large scale, and the last one "url" (for suspicious url detection) is large scale and high dimensional. In this experiment, for simplicity, we only compare the proposed SOFS algorithm with PET (due to its low time complexity) and FGM (due to its high accuracy).

Table V and Figure 6 show the experimental results of test accuracy and time cost of the three algorithms. We do not show the results of FGM on "url" as it is too slow. From the tables, we can observe that performance of SOFS is very close to or even better than that of FGM, especially when more features are selected. Both of these two algorithms outperform the baseline OFS algorithm PET. As to the time cost, the comparison of PET and SOFS on "news" and "rcv1" shows that the time cost of PET is more relevant to data dimension. An interesting phenomenon is that PET takes much more time when only 0.5% features are selected. This is because the PET algorithm converges much slower with a few features. In other words, the PET algorithm has to update its model frequently. FGM in this scenario is the most computationally expensive algorithm with even more than an order of magnitude difference. Besides, the time cost increases rapidly with more features selected. We can see the significant advantage of SOFS in these three high dimensional or large scale datasets. In real scenarios, it will be much



(a) news (b) rcv1

(c) news (d) rcv1

Fig. 6. Test accuracy and time cost versus number of selected features on the "news" and "rcv1" dataset.

more efficient to run the proposed FS algorithms multiple times on training datasets to achieve higher accuracy than running such computationally expensive batch learning algorithms.

Above all the experimental results and analysis, we find that the proposed OFS algorithm is able to outperform existing online methods and is comparable to state-of-the-art batch learning methods. However, computational efficiency of SOFS is significantly superior to both the online and batch FS algorithms, which capacitates the proposed SOFS to handle large scale and ultra-high dimensional data in real-world applications.

5. CONCLUSION AND FUTURE WORK

In this article, we addressed the challenge of FS for large-scale ultra-high-dimensional sparse data, which aims to select a small fixed number of relevant features. We presented a novel SOFS algorithm. In contrast to existing OFS algorithms with linear computational complexity on total feature dimensions, the complexity of our new algorithm is significantly reduced to be linearly dependent on number of non-zero features of each sample. We extensively evaluated empirical performance of the proposed algorithm by comparing with both online and batch state-of-the-art FS algorithms on both synthetic and real datasets, from medium scale to large scale. The promising results showed that our algorithm not only achieved highly competing prediction accuracy to the state-of-the-art batch FS algorithms, but also significantly improved computational efficiency, making our algorithm practical for handling large scale data with ultra-high dimensionality.

Despite the encouraging results, the existing solution for OFS could be further improved for future work. As we observed in experiments, prediction accuracy is worse than batch learning algorithms when only a small fraction of features are selected. We can explore more informativeness from data to enhance existing OFS algorithms with extremely few features. Another work is to adaptively select a number of features. Currently, we set the number of features manually. In industrial scenarios, the desirable case is that a FS system can receive data and then output the most compact and accurate model by the system itself. Besides, this paper focuses on the selection of distinctive features. Recently, researchers have explored structured knowledge based features to improve the downstream applications, such as text classification [Wang et al. 2016] and clustering [Wang et al. 2015]. How to select structured features efficiently and effectively is still under-explored.

REFERENCES

V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos. 2015. Recent advances and emerging challenges of feature selection in the context of big data. *Knowledge Based System* 86, C (Sept. 2015), 33–45.

Verónica Bolón-Canedo, Noelia Sánchez-Maroño, and Amparo Alonso-Betanzos. 2013. A review of feature selection methods on synthetic data. *Knowledge and Information Systems* 34, 3 (2013), 483–519.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *The Journal of Machine Learning Research* 7 (2006), 551–585.

Koby Crammer, Mark Dredze, and Alex Kulesza. 2009a. Multi-class confidence weighted algorithms. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP'09)*. Association for Computational Linguistics, Stroudsburg, PA, 496–504.

Koby Crammer, Mark Dredze, and Fernando Pereira. 2009b. Exact convex confidence-weighted learning. In *Advances in Neural Information Processing Systems (NIPS'09)*. Curran Associates, Inc., 345–352.

Koby Crammer, Alex Kulesza, and Mark Dredze. 2013. Adaptive regularization of weight vectors. *Machine Learning* 91, 2 (May 2013), 155–187.

Yi Ding, Peilin Zhao, Steven C. H. Hoi, and Yew-Soon Ong. 2015. Adaptive subgradient methods for online AUC maximization. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'15)*. AAAI Press, 2568–2574.

Mark Dredze, Koby Crammer, and Fernando Pereira. 2008. Confidence-weighted linear classification. In *Proceedings of the 25th International Conference on Machine Learning (ICML'08)*. ACM, New York, NY, 264–271.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12 (2011), 2121–2159.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. 2007. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. (2007). Retrieved from http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research* 9 (2008), 1871–1874.

Xingyu Gao, Steven C. H. Hoi, Yongdong Zhang, Ji Wan, and Jintao Li. 2014. SOML: Sparse online metric learning with application to image retrieval. In *Proceedings of the T28th AAAI Conference on Artificial Intelligence (AAAI'14)*. AAAI Press, Quebec City, Canada, 1206–1212.

Karen Glocer, Damian Eads, and James Theiler. 2005. Online feature selection for pixel classification. In *Proceedings of the 22th International Conference on Machine learning (ICML'05)*. ACM, New York, NY, 249–256.

Steven C. H. Hoi, Jialei Wang, Peilin Zhao, and Rong Jin. 2012. Online feature selection for mining big data. In *Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*. ACM, 93–100.

Steven C. H. Hoi, Jialei Wang, and Peilin Zhao. 2014. LIBOL: A library for online learning algorithms. *The Journal of Machine Learning Research* 15 (2014), 495–499. http://LIBOL.stevenhoi.org.

Jianping Hua, Waibhav D. Tembe, and Edward R. Dougherty. 2009. Performance of feature-selection methods in the classification of high-dimension data. *Pattern Recognition* 42, 3 (March 2009), 409–424.

Hao Huang, Shinjae Yoo, and Shiva Prasad Kasiviswanathan. 2015. Unsupervised feature selection on data streams. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM'15)*. ACM, New York, NY, 1031–1040.

Feng Jiang, Yuefei Sui, and Lin Zhou. 2015. A relative decision entropy-based feature selection approach. *Pattern Recognition* 48, 7 (July 2015), 2151–2163.

Sheng-yi Jiang and Lian-xi Wang. 2016. Efficient feature selection based on correlation measure between continuous and discrete features. *Information Processing Letters* 116, 2 (February 2016), 203–215.

Ron Kohavi and George H. John. 1997. Wrappers for feature subset selection. *Artificial Intelligence* 97, 1 (1997), 273–324.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 1097–1105.

Hoai An Le Thi, Xuan Thanh Vo, and Tao Pham Dinh. 2014. Feature selection for linear SVMs under uncertain data: Robust optimization based on difference of convex functions algorithms. *Neural Networks* 59 (November 2014), 36–50.

Fachao Li, Zan Zhang, and Chenxia Jin. 2016. Feature selection with partition differentiation entropy for large-scale data sets. *Information Sciences* 329, C (February 2016), 690–700.

Qing Li, Zhaofan Qiu, Ting Yao, Tao Mei, Yong Rui, and Jiebo Luo. 2017. Learning hierarchical video representation for action recognition. *International Journal of Multimedia Information Retrieval* 6, 1 (2017), 85–98.

Huan Liu and Lei Yu. 2005. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering* 17, 4 (2005), 491–502.

Jing Lu, Steven C. H. Hoi, Jialei Wang, and Peilin Zhao. 2013. Second order online collaborative filtering. In *Proceedings of the JMLR Workshop and Conference Proceedings (ACML'13), November 13–15, Canberra*, Vol. 29. 325–340.

Justin Ma, Alex Kulesza, Mark Dredze, Koby Crammer, Lawrence K. Saul, and Fernando Pereira. 2010. Exploiting feature covariance in high-dimensional online learning. In *Proceedings of the Artificial Intelligence and Statistics*. 493–500.

Sebastián Maldonado, Richard Weber, and Fazel Famili. 2014. Feature selection for high-dimensional class-imbalanced data sets using support vector machines. *Information Sciences* 286 (December 2014), 228–246.

Vijay Pappu, Orestis P. Panagopoulos, Petros Xanthopoulos, and Panos M. Pardalos. 2015. Sparse proximal support vector machines for feature selection in high dimensional datasets. *Expert Systems with Applications* 42, 23 (December 2015), 9183–9191.

Hanchuan Peng, Fuhui Long, and Chris Ding. 2005. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, 8 (2005), 1226–1238.

Simon Perkins and James Theiler. 2003. Online feature selection using grafting. In *Proceedings of the 20th Annual International Conference on Machine Learning (ICML'03)*. ACM, New York, NY, 592–599.

Sergio Ramírez-Gallego, Iago Lastra, David Martínez-Rego, Verónica Bolón-Canedo, José Manuel Benítez, Francisco Herrera, and Amparo Alonso-Betanzos. 2017. Fast-mRMR: Fast minimum redundancy maximum relevance algorithm for high-dimensional big data. *International Journal of Intelligent Systems* 32, 2 (2017), 134–152.

Frank Rosenblatt. 1958. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* 65, 6 (1958), 386.

K. Simonyan and A. Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Mingkui Tan, Ivor W. Tsang, and Li Wang. 2014. Towards ultrahigh dimensional feature selection for big data. *Journal of Machine Learning Research* 15, 1 (2014), 1371–1429.

Chenguang Wang, Yangqiu Song, Ahmed El-Kishky, Dan Roth, Ming Zhang, and Jiawei Han. 2015. Incorporating world knowledge to document clustering via heterogeneous information networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15)*. ACM, Sydney, NSW, Australia, 1215–1224.

Chenguang Wang, Yangqiu Song, Haoran Li, Ming Zhang, and Jiawei Han. 2016. Text classification with heterogeneous information network kernels. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence (AAAI'16)*. AAAI Press, Phoenix, AZ, 2130–2136.

Dayong Wang, Pengcheng Wu, Peilin Zhao, Yue Wu, Chunyan Miao, and Steven CH Hoi. 2014. High-dimensional data stream classification via sparse online learning. In *Proceedings of the IEEE International Conference on Data Mining (ICDM'14)*. IEEE, 1007–1012.

Jialei Wang, Peilin Zhao, and Steven C. H. Hoi. 2012. Exact soft confidence-weighted learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML'12)*. 121–128.

Jialei Wang, Peilin Zhao, and Steven C. H. Hoi. 2016. Soft confidence-weighted learning. *ACM Transactions on Intelligent Systems and Technology* 8, 1 (2016), 15.

Jialei Wang, Peilin Zhao, Steven C. H. Hoi, and Rong Jin. 2014. Online feature selection and its applications. *IEEE Transactions on Knowledge and Data Engineering* 26, 3 (2014), 698–710.

Yong Wang, Tao Mei, Shaogang Gong, and Xian-Sheng Hua. 2009. Combining global, regional and contextual features for automatic image annotation. *Pattern Recognition* 42, 2 (2009), 259–266.

Xindong Wu, Kui Yu, Hao Wang, and Wei Ding. 2010. Online streaming feature selection. In *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*. ACM, New York, NY, 1159–1166.

Zenglin Xu, Rong Jin, Jieping Ye, Michael R. Lyu, and Irwin King. 2009. Non-monotonic feature selection. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML'09)*. ACM, New York, NY, 1145–1152.

Haiqin Yang, Michael R. Lyu, and Irwin King. 2013. Efficient online learning for multitask feature selection. *ACM Transactions on Knowledge Discovery from Data* 7, 2 (August 2013), 6:1–6:27.

Haiqin Yang, Zenglin Xu, Michael R. Lyu, and Irwin King. 2015. Budget constrained non-monotonic feature selection. *Neural Networks* 71, C (November 2015), 214–224.

Lei Yu and Huan Liu. 2003. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th International Conference on Machine Learning (ICML'03)*, Vol. 3. ACM, New York, NY, 856–863.

Yiteng Zhai, Yew-Soon Ong, and I. W. Tsang. 2014. The emerging "big dimensionality". *Computational Intelligence Magazine, IEEE* 9, 3 (August 2014), 14–26.

Zheng Zhao, Lei Wang, Huan Liu, and Jieping Ye. 2013. On similarity preserving feature selection. *IEEE Transactions on Knowledge and Data Engineering* 25, 3 (2013), 619–632.

Martin Zinkevich. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the International Conference on Machine Learning (ICML'03)*.