Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Efficient Online Model Adaptation by Incremental Simplex Tableau

Zhixian Lei,[1] **Xuehan Ye,**[2] **Yongcai Wang,**[2*] **Deying Li,**[2] **Jia Xu**[3]
[1] Department of Computer Sciences, Harward University, USA;
[2] Department of Computer Sciences, Renmin University of China, Beijing;
[3] Department of Computer Sciences, City University of New York, USA

Abstract

Online multi-kernel learning is promising in the era of mobile computing, in which a combined classifier with multiple kernels are offline trained, and online adapts to personalized features for serving the end user precisely and smartly. The online adaptation is mainly carried out at the end-devices, which requires the adaptation algorithms to be light, efficient and accurate. Previous results focused mainly on efficiency. This paper proposes an novel online model adaptation framework for not only efficiency but also optimal online adaptation.

At first, an online optimal *incremental simplex tableau (IST)* algorithm is proposed, which approaches the model adaption by linear programming and produces the optimized model update in each step when a personalized training data is collected. But keeping online optimal in each step is expensive and may cause over-fitting especially when the online data is noisy. A *Fast-IST* approach is therefore proposed, which measures the deviation between the training data and the current model. It schedules updating only when enough deviation is detected. The efficiency of each update is further enhanced by running IST only limited iterations, which bounds the computation complexity. Theoretical analysis and extensive evaluations show that Fast-IST saves computation cost greatly, while achieving speedy and accurate model adaptation. It provides better model adaptation speed and accuracy while using even lower computing cost than the state-of-the-art.

Introduction

Online adaptive learning is widely required in the era of mobile and wearable computing(Song et al. 2014), in which a classifier, generally trained offline, needs to adapt to the personalized features after being delivered to end users. The goal is to serve the end users precisely and smartly (Gu, Pung, and Zhang 2005).

Challenges are posed from three aspects: 1) the online model adaptation is mainly carried out at the end devices, requiring the adaptation algorithms to be efficient for energy efficiency etc. 2) The personalized online training data is generally noisy, challenging the adaptation speed and the risk of over-fitting. 3) The training data comes from multiple sources or in different metric spaces, requiring adaptive feature selection and weighting (Hong, Suh, and Kim 2009).

Model learning from multiple features of different metric spaces, was traditionally, mainly studied in offline feature selection (Baram 2005) (Huang and Chow 2005). Automatic and adaptive feature weighting was proposed by multi-kernel learning (MKL)(Sonnenburg et al. 2006) (Gönen and Alpaydin 2011), which learns an optimal linear or non-linear combination of a set of predefined kernels to reduce the bias of manually weighting.

The flexibility of adaptively multiple feature weighting by combining multiple kernels enables recent studies for online multiple kernel learning (OMKL) (Martins et al. 2011)(Hoi et al. 2012)(Wang et al. 2015). OMKL generally composes two online learning sub-problems: 1) *perception* (Freund and Schapire 1999) , to train a basic classifier (BC) for each given kernel; and 2) *combination*, which is to weight and combine the BCs to generate a combined, more accurate classifier. Since the training of the BCs and the optimization of the combining weights for BCs are generally coupled, which involves quadratic programming(Lanckriet et al. 2004) or semi-infinite linear programming (Sonnenburg et al. 2006)(Rakotomamonjy et al. 2007), OMKL is generally computation extensive, even in offline training. Therefore, seeking efficiency is the major focus of current OMKL algorithms.

A major approach for efficiency is to decouple the perception and the classifier combination steps by training the BCs offline. This reduces the online learning problem to only adapt the combination weights based on the online training data (Kembhavi et al. 2009) (Orabona et al. 2010) (Jin, Hoi, and Yang 2010). The key idea is to increase the weights for the BCs whose outputs coincide with the online collected training data, and to decrease if inconsistent (Chaudhuri, Freund, and Hsu 2009)(Kembhavi et al. 2009).

But a key challenge of focusing on efficiency is the lack of guarantee on the responding speed. Like in *hedge*(Chaudhuri, Freund, and Hsu 2009)(Freund and Schapire 1997), the model coefficients are updated only one time when a training data is applied. The accurate adaptation thus requires corrections by many training instances, leading to slow adaptation. This paper, instead, seeks efficient, speedy, and optimal online adapation. It at first proposes an incremental simplex tableau (IST) algorithm, which can produce the optimal lin-

*corresponding author: Yongcai Wang, ycw@ruc.edu.cn

ear weights in each step based on the online collected training data. It is in practice carried out by simple row and column operations in the simplex tableau (Schrijver 1986). The intuition is that the tableau saves the previous optimal state, and the new optimal is generally not far from the old optimal, which can be achieved quickly.

But keeping optimal in each step is expensive and not necessary, especially when the training data is noisy. A Fast-IST framework is therefore developed for efficiency and noise tolerance. It uses a mini-batch, fixed-iteration online adaptation strategy (Bottou 1998). The online training data are organized into batches. Only when the majority of samples in a batch are far from the current model, will the algorithm update the model by fixed-iteration IST, i.e., fixed steps of simplex operations. In this way, the frequency of updating is reduced; the possible wrong samples are filtered; and the cost in each updating is bounded. Analysis and evaluations by both simulations and experimental data showed that Fast-IST provided better classification accuracy and speedy responding than the current OMKL approaches(Freund and Schapire 1997)(Chaudhuri, Freund, and Hsu 2009), using even less computation costs.

The rest of the paper is organized as following. Problem model and preliminaries are introduced in Section 2. IST is introduced in Section 3. Fast-IST is introduced in Section 4. Property analysis and performance evaluations are summarized in Section 5. The paper is concluded with remarks in Section 6.

Problem model and Preliminaries

This paper considers a general application scenario of OMKL, where a classifier composed by linear combination of multiple basic classifiers (BC) have been offline trained. We want the model, i.e., the linear weights, can online adapt to personalized data when the classifier is used by a user. The goal is to make the adaptation quick and use less computation cost. Since the multi-class classification problem can be transformed to binary classification by constructing a binary tree, for simplicity of exposition, we focus on binary classification in this paper.

Considering personalized data is arriving in a stream $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots$, where $\mathbf{x}_i$ is the ith feature data and y_i is the corresponding class label: $y_i \in \{-1, 1\}$. The set of BCs is $\{G_j : G_j(\mathbf{x}) \in \{-1, 1\}, j \in [m]\}$ where $G_j(\mathbf{x_i})$ is a prediction of y_i. After receiving nth data, ideally we hope to find suitable combining weights $\{a_j : j \in [m]\}$ such that the combined classifier can produce better classification than the offline trained one. The combined classifier is:

$$G(x) = \text{sign}\left[\sum_{j=1}^{m} a_j G_j(\mathbf{x})\right] \tag{1}$$

where $[n]$ and $[m]$ denote $\{1, 2, \ldots n\}$ and $\{1, 2 \ldots, m\}$ respectively. Since all G_js are BCs offline trained, we assume G_j can classify x_i correctly with probability higher than 50%, which is more powerful than random prediction.

The model adaptation can be transformed into an optimization problem. We seek $\forall j \in [m], a_j \geq 0$ and $\sum_{j=1}^{m} a_j = 1$, such that the following objective is minimized:

$$\text{minimize: } \sum_{i=1}^{n} \epsilon_i \tag{2}$$

$$\text{subject to: } y_i \sum_{j=1}^{m} a_j G_j(\mathbf{x}_i) \geq -\epsilon_i \qquad \forall i \in [n]$$

$$\epsilon_i \geq 0 \qquad \forall i \in [n]$$

where $\{\epsilon_i : i \in [n]\}$ is the possible error of the combined new classifier G.

Note that, the optimal G from this linear programming should have less training error than any G_j, i.e., providing not worse prediction accuracy for $\mathbf{x}_i$, since we can always achieve $G = G_j$ by assigning $a_j = 1$ and $a_i = 0\ \forall i \in [m], i \neq j$.

Theorem 1. *If basic classifiers predicts independently and randomly, the prediction error of the optimal G decreases exponentially with m.*

Proof. Suppose $\forall j \in [m]$, G_j randomly predicts with correct probability $p_j > 1/2$. Then the expectation $\mathbb{E}[y_i G_j(\mathbf{x}_i)] = p_j - (1 - p_j) = 2p_j - 1$ and

$$\mathbb{E}\left[y_i \sum_{j=1}^{m} a_j G_j(\mathbf{x}_i)\right] = \sum_{j=1}^{m} a_j (2p_j - 1) \tag{3}$$

If we take $a_j = 1/m\ \forall j \in [m]$ and let $\sum_{j=1}^{m}(2p_j - 1)a_j = 2p^* - 1$, by Hoeffding's inequality, the probability for incorrect prediction on $(\mathbf{x}_i, y_i)$ is

$$\Pr\left[y_i \sum_{j=1}^{m} a_j G_j(\mathbf{x}_i) < 0\right] \leq e^{-\Omega(m(2p^*-1)^2)} \tag{4}$$

Thus, the theorem holds. □

This theorem provides some worst case analyses for our mechanism when all basic classifiers are independent and random. The effectiveness of the algorithm in these settings in general demonstrates a better performance in real life application.

The key problem for online learning is to adapt the model weights efficiently and correctly. This contains two folds: 1) an efficient online model adapation algorithm; 2) efficient online data preprocessing to schedule the model update to avoid high computation cost and over-fitting. Solutions to these two problems are presented as following:

Incremental Simplex Tableau (IST)

Because the linear optimization problem in (2) can be optimally solved by linear programming. An incremental linear programming approach is designed for both optimality and efficiency.

Simplex Tableau

For self-containing, the simplex tableau method and related operations are introduced in this part. For more information about simplex algorithm and related topics, please refer to (Schrijver 1986)

Simplex algorithm is a method for solving general linear programming problem, which finds a vertex $\mathbf{x}$ to minimize objective $\mathbf{u}^t\mathbf{x}$ from vertexes on a high dimensional convex polytope defined by $\{\mathbf{x} : \mathbf{D}\mathbf{x} \leq \mathbf{f}, \mathbf{x} \geq 0\}$. Since the optimal solution can always be found in the convex linear programming problem, the algorithm stops when no neighbor vertex have smaller $\mathbf{u}^t\mathbf{x}$ than the current vertex. The *dual simplex algorithm* can be considered as running simplex algorithm on the dual problem of the linear programming.

The simplex algorithm is often executed inside *simplex tableau*, which converts the problem into a standard form:

$$\begin{aligned} \text{minimize: } & \mathbf{u}^t\mathbf{x} \\ \text{subject to: } & \mathbf{D}\mathbf{x} + \mathbf{y} = \mathbf{f} \\ & \mathbf{x} \geq \mathbf{0} \\ & \mathbf{y} \geq \mathbf{0} \end{aligned} \tag{5}$$

Then it is written into a tableau representation:

$$\left[\begin{array}{cc|c} \mathbf{0} & -\mathbf{u}^t & \delta \\ \hline \mathbf{I} & \mathbf{D} & \mathbf{f} \end{array}\right] \tag{6}$$

which is the initial tableau for the problem, where $\mathbf{I}$ is the identity matrix and $\mathbf{0}$ is the all-zero matrix.

The simplex and dual simplex algorithm iteratively exchange one variable in $\mathbf{x}_B$ with one variable in $\mathbf{x}_N$ until obtaining the the optimal solution. The optimal solution is achieved when $\mathbf{u} \geq 0$ and $\mathbf{f} \geq 0$. The exchange is achieved by conducting pivot operations repeatedly.

Definition 1 (pivot operation)**.** *The pivot operation contains following two steps:*

1. *Exchange the columns of two variables to make the column of the variable in $\mathbf{x}_N$ inside the identity matrix.*
2. *Use Gaussian elimination to eliminate the off-diagonal terms in the arriving column inside the identity matrix and the first row to reconstruct the identity matrix.*

When it is not optimal, the simplex tableau has following two states and corresponding updating schemes:

- **FnO State:** If $\mathbf{f} \geq \mathbf{0}$ but $\mathbf{u} \ngeq \mathbf{0}$, the solution is feasible but not optimal. Suppose $u_k < 0$ and the variable corresponding to the column of u_k is $x_k \in \mathbf{x}_N$, Simplex exchanges x_k with one variable in $\mathbf{x}_B$ to make $u_k = 0$. Note that, it chooses suitable variable to leave $\mathbf{x}_B$ such that $\mathbf{f} \geq 0$ always holds. It repeats this process until $\mathbf{u} \geq 0$ to find the optimal solution.
- **OnF State:** If $\mathbf{u} \geq \mathbf{0}$ but $\mathbf{f} \ngeq \mathbf{0}$, the solution is optimal but not feasible. Then dual simplex algorithm can be used to adjust the simplex tableau to the optimal one. Suppose $f_k < 0$ and the variable in the identity matrix assigned by f_k is $x_k \in \mathbf{x}_B$. Then x_k is exchanged with one variable in $\mathbf{x}_N$ to make $f_k \geq 0$. It chooses suitable variable to enter $\mathbf{x}_B$ such that $\mathbf{u} \geq \mathbf{0}$ holds. The process is repeated until $\mathbf{f} \geq \mathbf{0}$ to find the optimal solution.

In Simplex, when choosing the suitable variable to exchange, if there are multiple choices, the rule is to choose the variable that can decrease δ to the largest extent. When using dual-simplex algorithm, the rule is to choose variable to increase δ to the largest extent. Since δ is the value of $\mathbf{c}^t\mathbf{x}$, in this way, the optimal solution can be achieved in the fastest way.

Incremental Simplex Tableau

The linear programming problem stated in (2)-(6) can be transformed into the standard linear programming form, whose initial simplex tableau can be represented as:

$$\left[\begin{array}{ccc|c} 0 & -\mathbf{1} & \mathbf{0} & 0 \\ \hline \mathbf{1} & \mathbf{0} & \mathbf{0} & 1 \\ \mathbf{A} & \mathbf{I} & -\mathbf{I} & \mathbf{0} \end{array}\right] \tag{7}$$

where

$$\mathbf{A} = \begin{bmatrix} y_1G_1(x_1) & \cdots & y_1G_m(x_1) \\ \vdots & \ddots & \vdots \\ y_nG_1(x_n) & \cdots & y_nG_m(x_n) \end{bmatrix} \tag{8}$$

After the offline training stage, suppose the optimal simplex tableau corresponding to the optimal solution has been obtained:

$$\left[\begin{array}{cc|c} \mathbf{0} & -\mathbf{u}^t & \delta \\ \hline \mathbf{I} & \mathbf{D} & \mathbf{f} \end{array}\right] \tag{9}$$

where $\mathbf{u} \geq 0$ and $\mathbf{f} \geq 0$. Then the tableau preserves all the offline learned knowledge. The optimal solution $\mathbf{x}_B = \mathbf{f}$, $\mathbf{x}_N = \mathbf{0}$ and the minimal value δ can be found.

Then in the online phase, it only needs to consider how to update the model by collecting online training data and how to delete old training data, to keep a moderate size of training set when using the simplex and dual simplex algorithms.

1) Update by New Training Data When new training data (x_{n+1}, y_{n+1}) is collected, new constraints are formed:

$$\begin{aligned} y_{n+1}\sum_{j=1}^{m} a_jG_j(x_{n+1}) + \epsilon_{n+1} - \lambda_{n+1} = 0 \\ \epsilon_{n+1} \geq 0 \\ \lambda_{n+1} \geq 0 \end{aligned} \tag{10}$$

Note that λ_i is the artificial variable to write the linear programming model to standard form. By adding the new constraints into the simplex tableau, the new simplex tableau becomes:

$$\left[\begin{array}{cccc|c} \mathbf{0} & 0 & -\mathbf{u}^t & -1 & \delta \\ \hline \mathbf{I} & \mathbf{0} & \mathbf{D} & \mathbf{0} & \mathbf{f} \\ \mathbf{d}_1^t & 1 & \mathbf{d}_2^t & -1 & 0 \end{array}\right] \tag{11}$$

where $\mathbf{d}_1$, $\mathbf{d}_2$ consists of $-y_{n+1}G_1(x_{n+1})$, $\cdots$, $-y_{n+1}G_m(x_{n+1})$ and zeros. Then Gaussian elimination is utilized to cancel $\mathbf{d}_1$ and obtain the standard form of the simplex tableau.

$$\left[\begin{array}{cccc|c} \mathbf{0} & 0 & -\mathbf{u}^t & -1 & \delta \\ \hline \mathbf{I} & \mathbf{0} & \mathbf{D}' & \mathbf{0} & \mathbf{f} \\ 0 & 1 & \mathbf{d}_2^t - \mathbf{d}_1^tD & -1 & -\mathbf{d}_2^t\mathbf{f} \end{array}\right] \tag{12}$$

CASE 1: If $-\mathbf{d}_2^t\mathbf{f} \geq 0$, the optimal solution is obtained immediately. The optimal weights remain the same. The newly added variables $\lambda_{n+1} = -\mathbf{d}_2^t\mathbf{f}$, $\epsilon_{n+1} = 0$ and λ_{n+1} are added into $\mathbf{x}_B$; ϵ_{n+1} is added into $\mathbf{x}_N$.

CASE 2: If $-\mathbf{d}_2^t\mathbf{f} < 0$, it is the only negative entry in the last column. Since in the new simplex tableau $\mathbf{u}'^t = (\mathbf{u}^t, 1) \geq 0$, the tableau is in **OnF state**, which can be updated by iterating the pivot operation in dual simplex algorithm until $-\mathbf{d_2}\mathbf{f} \geq 0$ and $\mathbf{u} \geq 0$ to achieve the new optimal solution.

2) Update by Deleting Past Data Instead of always increasing the training set, deleting scheme is also needed to keep a moderate size of training set in the tableau representation, which not only keeps the data freshness, but also benefits the computing efficiency.

Suppose $(\mathbf{x_1}, y_1)$ is to be deleted from the system, we need to remove both the constraint and the column variable data.

i. Remove a constraint: To remove the first constraint, we only need to remove ϵ_1 from the objective, because if there is no ϵ_1 in the objective, the first constraint $y_1 \sum_{j=1}^m a_j G_j(\mathbf{x_1}) + \epsilon_1 - \lambda_1 = 0$ can be always satisfied by choosing $\epsilon_1 - \lambda_1 = -y_1 \sum_{j=1}^m a_j G_j(\mathbf{x_1})$. Therefore the entry corresponding to ϵ_1 in the objective row should be increased by 1 to maintain the relation $\mathbf{c}^t\mathbf{x} - \mathbf{u}^t\mathbf{x}_N = \delta$ in the simplex tableau. When the constraint does not confine the problem, it is identical as the constraint is removed.

i.1 CASE 1 : If ϵ_1 is currently the non-basic solution and $\mathbf{u} \geq \mathbf{0}$ is satisfied, the simplex tableau is still optimal and the assignment of the optimal solution does not change.

i.2 CASE 2 : If the entry corresponding to ϵ_1 becomes negative, since $\mathbf{f} \geq \mathbf{0}$ still holds. The tableau is in an **FnO state**, and this new optimal solution can be obtained by iterating pivot operations in the simplex algorithm until $\mathbf{u} \geq 0$.

ii. Remove the variables: Not only the constraints but also the corresponding variables need to be removed to keep the size of the simplex tableau from always growing. It is required to remove the column corresponding to ϵ_1 and λ_1. Since the column corresponding to ϵ_1 and the column corresponding to λ_1 are always summed to 0 in simplex tableau, by elementary row operation, $\epsilon_1 - \lambda_1$ can be eliminated simultaneously using $y_1 \sum_{j=1}^m a_j G_j(\mathbf{x_1})$. Also all zero rows should appear in the standard simplex tableau after canceling ϵ_1 and λ_1. Since one constraint will be linear dependent to other constraints, by conducting Gaussian elimination, we can remove this row and make the size of the simplex tableau always consistent to the size of the linear programming problem.

IST keeps the online optimal classifier model for the current training dataset. But a notable problem of IST is that it updates the model to the optimal state when any new training data is arrived, which is expensive and not necessary especially when the online training data is noisy. Another limitation is that the worst-cast number of iterations for achieving optimal is not determined. We therefore proposed a FAST-IST framework for efficiency and robustness.

FAST-IST Framework

FAST-IST is composed by 1) a data processing scheme to to schedule update; 2) model update by IST with a fixed number of iterations to bound computation cost, i.e., runs only a fixed number of pilot operations in each update.

Data Processing to Control Update

Given data stream $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$, we organize successive l data points $\{(\mathbf{x}_i, y_i), i \in [l]\}$ into a batch $\mathbf{S}$ of length l. We want to use these l samples to measure the derivation between the current model and the expected model of the training data. The idea comes from majority voting(Ao et al. 2016) (Penrose 1946)(Lam and Suen 1997). If most of data points in $\mathbf{S}$ can be fitted well by the current model, with high probability the minority unfitted samples in $\mathbf{S}$ come from noise. So the model is not necessary to update. But when the majority of data points violate the current model, the current model should be updated.

To measure how well the current model fits the data point $(\mathbf{x}, y)$, we can compare the error corresponding to the data point with the average error of the model. More specifically, suppose the current optimal model is $\mathbf{a}^t = (a_1, a_2, \ldots, a_m)$ and the average error is $\epsilon = \frac{1}{n}\sum_{i=1}^n \epsilon_i$. If

$$-y \sum_{j=1}^{m} a_j G_j(\mathbf{x}) \geq \epsilon \tag{13}$$

it indicates that the sample data can not be well fitted by the current model. Otherwise, the model is measured to fit the data well. Using this measure of fitness, the data filtering algorithm is as follows:

1. Divide stream into mini-batch $\{(\mathbf{x}_i, y_i), i \in [l]\}$ of size l
2. For all samples in $\{(\mathbf{x}_i, y_i), i \in [l]\}$, if $-y_i \sum_{j=1}^m a_j G_j(\mathbf{x}_i) \geq \epsilon$, count $(\mathbf{x}_i, y_i)$ as unfitted data point
3. If number of unfitted data point is less than threshold T, remove $\{(\mathbf{x}_i, y_i), i \in [l]\}$ from the stream.
4. Otherwise, update the model by unfitted batch data.

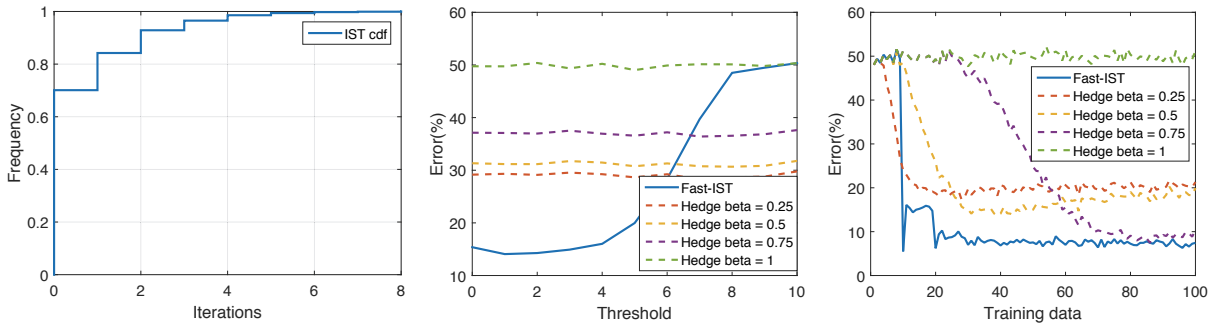
Fixed-Iteration IST

Since the training data may be noisy, it is not necessary to always pursue the optimal model to fit the data. Thus, fixed-iteration IST is used in model update, in which the number of iterations in Simplex is set to a constant C.

Deleting data: When deleting past data, since simplex algorithm searches solution in the feasible region of the problem, the non-optimal solution is still feasible which can be used directly to form the combined classifier G.

Adding data: When adding new data, the dual simplex algorithm is applied to find the optimal solution. As dual simplex algorithm searches solution in the feasible region of the dual problem, the non-optimal solution may not satisfy all constraints of the problem. If there exists $a_j < 0$ in the non-optimal solution, we force $a_j = 0$ and normalize $\mathbf{a}$ to make the weights satisfy the constraints.

Theorem 2. *The computation complexity of Fixed-Iteration IST is $O(n^2)$, where n is the size of the online data set.*

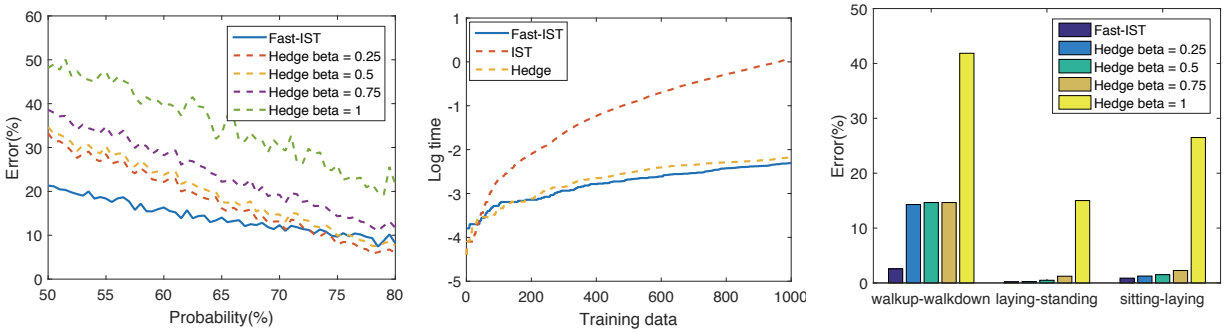


(a) CDF graph of number of iterations of IST

(b) Percent of error prediction by Fast-IST as a function of threshold in window

(c) Percentage of error prediction by Fast-IST as a function of the number of training data

Figure 1: Evaluation of parameters and response speed



(a) Percent of error prediction by Fast-IST as a function of the correct prediction probability of the basic classifiers

(b) Running time of Fast-IST as a function of the number of training data

(c) Percent of error prediction by Fast-IST in different class partitions

Figure 2: Evaluation of accuracy and efficiency

Proof. The process of adding new data and deleting past data is symmetric and they have the same computational complexity. When the system changes, a pivot operation will visit every entry inside the columns corresponding to $\mathbf{x}_N$. The size of the simplex tableau is $n \times (m + 2n)$ and there are $m + n$ columns corresponding to $\mathbf{x}_N$. So it takes $O(Cn(m + n))$ steps for one update in Fixed-Iteration IST. Since C is a constant and $m < n$, the computational complexity for once update is $O(n^2)$. □

As the average complexity for simplex algorithm is $O(n^3)$, The fixed iteration IST gives a $\Omega(n)$ speed up in each iteration. If we further consider the saving from data processing we will get an improvement of order $\Omega(l)$ for a carefully selected batch size l. There is a potential problem when n is large since every time we use fixed iteration IST to approximate optimal solution, the error is accumulated when n is large. This can undermine the efficiency of fixed iteration IST for later online training therefore we control the number of training data n by alternately adding and deleting data to keep fast updating.

Performance evaluation

Performance evaluations of FAST-IST were conducted by both simulations and actual experimental data. Efficiency and accuracy are evaluated as the key performance for online adaptation.

Settings of Simulation

In simulations, basic classifiers G_j are enumerated by random binary predictors with prediction accuracy p_j. Four aspects are mainly evaluated including 1) parameter impacts in Fast-IST; 2)responding speed; 3) accuracy; and 4) running efficiency. The parameters of Fast-IST include the number of IST iterations, and the threshold for triggering update. Response speed is assessed by the number of online data needed for achieving stable a_j.

The simulation uses m basic classifiers and n online training data. The correct probability of m basic classifiers are set uniformly distributed from 55% to 80% to simulate BCs of different accuracy. In online phase, the correct probability of each BC p_j varies to simulate the impacts of the personalized features to the classifier model.

Fast-IST is compared with *hedge algorithm*(Chaudhuri, Freund, and Hsu 2009), which is widely used in online

learning especially in online multi-kernel learning model. In hedge algorithm, if some classifier makes a wrong prediction, when next sample of data comes, the weight of this classifier will be penalized by a ratio $0 < \beta < 1$. And the hedge algorithm predicts by weighting and summarizing all the basic classifiers' results.

Parameter Impacts to Fast-IST

Impacts of number of iterations At first, we verified how the number of iterations in Fast-IST will affect the performance. We investigated the number of iterations for achieving the optimal in IST. The result, i.e., the distribution of iteration number in IST for achieving the optimal update is plotted in Fig.1(a), which is the cumulative distribution function (CDF) of the iteration number. It can be seen that IST generally needs only a few iterations to update the model to the new optimal state. This indicates the validity of bounding the iteration times in Fast-IST, which will not much affect the performance.

The threshold to trigger model update . In Fast-IST, only when the numbers of wrong predictions in a batch exceeds a threshold T, will a new model update be triggered. How dose T affect the prediction accuracy of the combined classier after applied 100 instances of training data is shown Fig.1(b). In simulations the batch size is set to 10. It can be seen that, the optimal threshold is 2 or 3 when the batch size is 10. Setting higher threshold, although can reduce the triggering of update, but leads to much worse model accuracy.

Responding Speed

Responding speed is a key performance of online adaptation algorithms. Fig.1(c) compares the response speed between Fast-IST and hedge of different parameters. It can be seen that Fast-IST adjusts a_j more speedy than hedge. Hedge algorithms with small β, e.g. 0.25 and 0.5, also adapt quickly, because smaller β adjusts weights more severely than bigger β. But hedge with smaller β generally provides worse accuracy than Fast-IST. Fast-IST converges quickly, and there are two obvious turnings points. The reason is that the batch length is 10. The turning points are where Fast-IST starts to correct the model weights.

Accuracy

Fig.2(a) compares Fast-IST and hedge algorithm on the prediction accuracy. In evaluation, the correct prediction probabilities of all basic classifiers are varying from 50% to 80%. It can be seen that Fast-IST have much less number of error prediction than hedge algorithm for all the β.

Computation Efficiency

The running time of Fast-IST are compared with hedge algorithm for 1000 online data. For clear comparison, we plot log results of running time in Fig.2(b). It claims Fast-IST works a little faster than hedge algorithm and IST works highly more slowly than the other two algorithms. The reason why Fast-IST is even faster than hedge is that after t, i.e., the number of training data, is larger than 100, update is rarely triggered in Fast-IST, since the model has fit the training data well. But hedge still updates frequently.

Evaluation by Activity Recognition Dataset

Experiment Settings Human Activity Recognition Using Smartphones Data Set from open data set UCI (Lichman 2013) is used to evaluate Fast-IST in actual experiment. The database is composed of 7352 samples from 30 users with 561 types of sensor data from smartphones and 6 classes, <walking, walking-up, walking-down, sitting, standing, laying>. We separately calculated error percent <walking-up, walking-down>, <laying, standing> and <sitting, standing>, which are hard to be classified. 187 basic KNN classifiers are chosen, each of which is trained on 3 types of sensor data.

The prediction accuracy of Fast-IST and hedge algorithm for <walking-up, walking-down>, <laying, standing> and <sitting, standing> are calculated and plotted in Fig.2(c). The prediction accuracy distribution of 187 basic classifiers for three class partitions, i.e. mean, max, min, are shown in Table.1. It can be seen that Fast-IST and hedge algorithm indeed give much more accurate predictions than the basic classifiers. Especially, Fast-IST provides much better prediction accuracy than hedge algorithms in all the three classification problems. In classifying some hard cases such as walking up and walking down, Fast-IST provides significant accuracy improvement.

	mean	max	min
walkup-walkdown	68.3253	86.0806	47.6190
laying-standing	63.0685	100	48.8358
sitting-laying	59.2011	99.0560	48.3952

Table 2: Accuracy of basic classifiers (%)

Conclusion

This paper has investigated online learning problem for adaptively personalizing learning model, where multiple classifier combination is considered in particular. A simplex based online optimizing algorithm is proposed for both efficient and performance-guaranteed online model updating. We give the general IST algorithm and improved Fast-IST algorithm by proper data filtering and iteration bounding. Analysis shows that IST can guarantee the online optimal and Fast-IST provides near-optimal update with higher efficiency. The applications, specification, and further optimization of IST and Fast-IST for particular problems, and the development of IST for batch updates will be investigated in future studies.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China Grant No. 11671400, 61672524; the Fundamental Research Funds for the Central University, and the Research Funds of Renmin University of China, 2015030273; National Science Foundation of US awards Computing and Communication Foundations No.1565264 and Computer and Network Systems No.1618026.

References

Ao, B.; Wang, Y.; Yu, L.; Brooks, R. R.; and Iyengar, S. S. 2016. On precision bound of distributed fault-tolerant sensor fusion algorithms. *ACM Comput. Surv.* 49(1):5:1–5:23.

Baram, Y. 2005. Learning by kernel polarization. *Neural Computation* 17(6):1264–1275.

Bottou, L. 1998. Online learning and stochastic approximations. *On-line learning in neural networks* 17(9):142.

Chaudhuri, K.; Freund, Y.; and Hsu, D. J. 2009. A parameter-free hedging algorithm. In *Advances in neural information processing systems*, 297–305.

Freund, Y., and Schapire, R. E. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* 55(1):119 – 139.

Freund, Y., and Schapire, R. E. 1999. Large margin classification using the perceptron algorithm. *Mach. Learn.* 37(3):277–296.

Gönen, M., and Alpaydin, E. 2011. Multiple kernel learning algorithms. *J. Mach. Learn. Res.* 12:2211–2268.

Gu, T.; Pung, H. K.; and Zhang, D. Q. 2005. A serviceoriented middleware for building contextaware services. *Journal of Network and Computer Applications* 28(1):1 – 18.

Hoi, S. C. H.; Jin, R.; Zhao, P.; and Yang, T. 2012. Online multiple kernel classification. *Machine Learning* 90(2):289–316.

Hong, J.-y.; Suh, E.-h.; and Kim, S.-J. 2009. Context-aware systems: A literature review and classification. *Expert Systems with Applications* 36(4):8509–8522.

Huang, D., and Chow, T. W. 2005. Effective feature selection scheme using mutual information. *Neurocomputing* 63:325 – 343. New Aspects in Neurocomputing: 11th European Symposium on Artificial Neural Networks.

Jin, R.; Hoi, S.; and Yang, T. 2010. Online multiple kernel learning: Algorithms and mistake bounds. In Hutter, M.; Stephan, F.; Vovk, V.; and Zeugmann, T., eds., *Algorithmic Learning Theory*, volume 6331 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 390–404.

Kembhavi, A.; Siddiquie, B.; Miezianko, R.; McCloskey, S.; and Davis, L. 2009. Incremental multiple kernel learning for object recognition. In *Computer Vision, 2009 IEEE 12th International Conference on*, 638–645.

Lam, L., and Suen, S. 1997. Application of majority voting to pattern recognition: an analysis of its behavior and performance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 27(5):553–568.

Lanckriet, G. R. G.; De Bie, T.; Cristianini, N.; Jordan, M. I.; and Noble, W. S. 2004. A statistical framework for genomic data fusion. *Bioinformatics* 20(16):2626–2635.

Lichman, M. 2013. UCI machine learning repository.

Martins, A. F. T.; Smith, N. A.; Xing, E. P.; Aguiar, P. M.; and Figueiredo, M. A. 2011. Online learning of structured predictors with multiple kernels. In *AISTATS*, 507–515.

Orabona, F.; Fornoni, M.; Caputo, B.; and Cesa-Bianchi, N. 2010. Om-2: An online multi-class multi-kernel learning algorithm luo jie. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, 43–50.

Penrose, L. S. 1946. The elementary statistics of majority voting. *Journal of the Royal Statistical Society* 109(1):53–57.

Rakotomamonjy, A.; Bach, F.; Canu, S.; and Grandvalet, Y. 2007. More efficiency in multiple kernel learning. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, 775–782. New York, NY, USA: ACM.

Schrijver, A. 1986. *Theory of Linear and Integer Programming*. New York, NY, USA: John Wiley & Sons, Inc.

Song, L.; Wang, Y.; Yang, J. J.; and Li, J. 2014. Health sensing by wearable sensors and mobile phones: A survey. In *e-Health Networking, Applications and Services (Healthcom), 2014 IEEE 16th International Conference on*, 453–459.

Sonnenburg, S.; Rätsch, G.; Schäfer, C.; and Schölkopf, B. 2006. Large scale multiple kernel learning. *J. Mach. Learn. Res.* 7:1531–1565.

Wang, Z.; Fan, Q.; Ke, S.; and Gao, D. 2015. Structural multiple empirical kernel learning. *Information Sciences* 301:124 – 140.