Full Parameter Time Complexity (FPTC): A Method to Evaluate the Running Time of Machine Learning Classifiers for Land Use/Land Cover Classification

Xiaorou Zheng, Jianxin Jia, Shanxin Guo, Jinsong Chen, Luyi Sun, Yingfei Xiong, and Wenna Xu

***Abstract*—In emergency responses to natural disasters, actionable information provided by remote sensing images is crucial to help emergency managers become aware of the situation and assess the magnitude of the damage. Without the accurate prediction of time consumption, choosing an algorithm for land use/land cover (LULC) classification under these emergency circumstances could be blind and subjective. Here, we proposed a full parameter time complexity (FPTC) analysis and the corresponding coefficient ω to estimate the actual running time of the LULC classification without actually running the code. The FPTC of five general algorithms is derived in this article. After derivation, the FPTC of k-nearest neighbors (kNN) is $F(nv + n\log_2 u)$, the FPTC of logistic regression (LR) is $F(Qm^2vn)$, the FPTC of classification and regression tree (CART) is $F((m+1)nv\log_2 n)$, the FPTC of random forest (RF) is $F(s(m+1)nv\log_2 n)$, and the FPTC of support vector machine (SVM) is $F(m^2Qv\,(n+k))$. The results show a strong linear relationship between the actual running time and FPTC [R-squared: kNN (0.991), LR (0.997), CART (0.999), RF (1.000), and SVM (0.999)], with different data size. The average root-mean-squared error between the real running time and the estimated running time is 3.34 s, which demonstrates the effectiveness of FPTC. Combining FPTC with the corresponding coefficient ω, the running time of the classification can be precisely predicted, which will help emergency managers quickly choose algorithms in response to natural disasters with available remote sensing data and limited time.**

Manuscript received November 2, 2020; revised December 24, 2020; accepted January 4, 2021. Date of publication January 8, 2021; date of current version February 8, 2021. This work was supported in part by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDA19030301, in part by the National Key Research and Development Program of China under Grant 2017YFB0504203, in part by the Natural Science Foundation of China Project under Grant 41601212, Grant 41801360, Grant 41771403, and Grant 41801358, and in part by the Fundamental Research Foundation of Shenzhen Technology and Innovation Council under Grant KCXFZ202002011006298. *(Corresponding author: Shanxin Guo.)*

Xiaorou Zheng, Yingfei Xiong, and Wenna Xu are with the Center for Geo-Spatial Information, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China and also with the University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: xiaorou.zheng@siat.ac.cn; yf.xiong@siat.ac.cn; wn.xu@siat.ac.cn).

Jianxin Jia is with the Center for Geo-Spatial Information, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China, with the Shenzhen Engineering Laboratory of Ocean Environmental Big Data Analysis, and Application, Shenzhen 518055, China, and also with the Department of Photogrammetry, and Remote Sensing, Finnish Geospatial Research Institute, 02430 Krikkonummi, Finland (e-mail: jx.jia@siat.ac.cn).

Shanxin Guo, Jinsong Chen, and Luyi Sun are with the Center for Geo-Spatial Information, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China, and also with the Shenzhen Engineering Laboratory of Ocean Environmental Big Data Analysis, and Application, Shenzhen 518055, China (e-mail: sx.guo@siat.ac.cn; js.chen@siat.ac.cn; ly.sun@siat.ac.cn).

Digital Object Identifier 10.1109/JSTARS.2021.3050166

***Index Terms*—Algorithm running time, full parameter time complexity (FPTC), land use/land cover (LULC) classification, Sentinel-2a, traditional time complexity (TTC).**

NOMENCLATURE

$\boldsymbol{T_r} \in \mathbb{R}^{n\times v}$	Training dataset
$\boldsymbol{T_e} \in \mathbb{R}^{z\times v}$	Test dataset
n	The number of total samples for $\boldsymbol{T}_r$
v	The number of bands/features
m	The number of categorizations
b	The number of total samples for $\boldsymbol{T}_e$
t'	The running time
C	The regularization parameter
ω	The constant coefficient
$\boldsymbol{\theta}'$	The model parameters
u	The number of object group in kNN
λ	The learning rate
s	The number of trees in the random forest
Q	The total iterations
k	The number of support vectors
γ	The kernel parameters in SVM

I. INTRODUCTION

ASSESSMENTS of natural disasters and risk are the foundation of decision-making processes for a wide variety of actors from the public to government emergency managers. Quickly quantifying damage and expected future losses is usually the first step to becoming aware of the current situation [1]–[3]. The land use/land cover (LULC) products from remote sensing imagery can provide first-hand information for this purpose [4]–[6]. As a result that this decision-making process is usually urgent, choosing an appropriate classification algorithm to achieve this goal with limited time and resources can be challenging [7]. In addition to classification accuracy, the actual time consumption of the algorithm is another aspect that needs to be carefully evaluated before running the task [4], [7], [8]. Without an accurate prediction of time consumption, choosing an algorithm for LULC classification under these emergency circumstances could be blind and subjective.

In general, the methods to estimate the time consumption of a classification task can be divided into two categories: 1) sampled-data-based methods and 2) time-complexity-based methods [9], [10]. The first category involves estimating based

on running the program and launching the time-calculated function on the sampled dataset. These methods hold an assumption that the actual running time between the sample and the whole dataset can be simplified by a linear or nonlinear relationship. Although these methods are used in various studies [8], [10], [11], the drawbacks of these methods are that 1) this linear or nonlinear relationship highly depends on the hardware, which cannot be generalized across different computing environments, and 2) the effect of the different parameters of the algorithm (both operational and hidden parameters) is treated as a black box. The influence of these parameters remains unknown.

The second category involves estimating based on a time complexity analysis. The commonly used asymptotic time complexity belongs to this category, usually referred to as traditional time complexity (TTC) [9]. TTC is a function of input size (e.g., the number of samples), which measures the computational complexity with the input size increase under the iterations of unit operation (e.g., addition or multiplication). The time of each unit operation is assumed to have the same value, so the iterations can be estimated as proportional to the running time [9], [12]. In the process of calculating the TTC, many low-order details have been ignored. For instance, when we calculate TTC for a function $f(n) = an^2 + bn + c$ (where n represents the data size), only $f(n) = n^2$ is of concern [9], [12], [13]. As a result, $O(n^2)$ is used to estimate the upper boundary of the time complexity of this function. The purpose of the TTC is to capture the acceleration of the running time as an increase in the data size (*N*) at the theoretical level; this can hardly be used for the accurate prediction of the running time, especially for remote sensing LULC classification tasks, where the time consumption not only relates to the data size (*N*) but also to other parameters (e.g., the number of available bands and the number of support vectors). How to consider the influence of these parameters to predict the overall time consumption remains a challenge.

In fact, without considering the physical discrepancy between the different platforms (such as CPU and GPU), the time consumption of a classification algorithm can be influenced by 1) the date size, 2) the number of classes, 3) the number of bands/features, 4) the iteration structure of the algorithm, 5) the operational parameters of the algorithm, such as the number of trees in random forest, and 6) the hidden parameters of the algorithm, such as the number of support vectors. All these components affect the actual time consumption of the algorithm in different ways via unknown mechanisms. Determining how the contribution of each component can be quantified is key to predicting the actual time that will be consumed.

To address the abovementioned issues, we propose full parameter time complexity (FPTC), which takes all time-consuming parameters into account. The FPTC of five general algorithms—*k*-nearest neighbors (kNN) [14], logistic regression (LR) [15], classification and regression tree (CART) [16], random forest (RF) [17], and support vector machine (SVM) [18]—is derived in this article. We defined a coefficient ω to model the physical discrepancy between different platforms for different classifiers. To test the effectiveness of FPTC and the corresponding coefficient ω, the Xinjiang Uygur Autonomous Region, China, and the Sentinel-2 A dataset were chosen as a case study. The results show that the running time of the classification task can be precisely predicted by combining FPTC with coefficient ω. These will help emergency managers make quick decisions in response to natural disasters. Our contribution can be summarized as follows.

1) We propose a method to quantitatively evaluate the time efficiency of a machine learning classifier (FPTC) and derive the FPTC of five general algorithms: kNN, LR, CART, RF, and SVM.
2) To predict the time consumption, we propose the coefficient ω, which is used to establish the relationship between running time and FPTC. The coefficient ω can be easily obtained with the pre-experiment with a small sampled dataset under different computing environments.
3) For the parameters that cannot be estimated before running the algorithm, we analyze the relationship between these parameters and those easily obtained hyperparameters to predict the actual running time without actually running the algorithm.

The remainder of this article is organized as follows. In Section II, FPTC is defined, and the corresponding mathematical derivation is described in detail. Sections III and IV present the materials and the experimental result of the Xinjang dataset. Section V concludes with a summary.

II. Full Parameter Time Complexity

A. Definition

FPTC is defined as two components. One is $F(n, m, v, \boldsymbol{\theta}')$, the algorithm-related part, which can be derived based on the structural analysis for one particular classifier. This part is a function of n, m, v and $\boldsymbol{\theta}'$, where n represents the sample size, m represents the number of targeted classes, v represents the number of bands/features of the remote sensing image, and $\boldsymbol{\theta}'$ represents a collection of parameters related to the algorithm. It should be noticed that the $\boldsymbol{\theta}'$ can be different for different algorithms. For instance, $\boldsymbol{\theta}'$ of the FPTC in kNN consists of the number of nearest neighbors u, while the $\boldsymbol{\theta}'$ of the FPTC in SVM consists of the number of iterations Q and the number of support vectors k. The second component is the coefficient ω, which is a physically related part reflecting the computing environmental factors, such as the speed of the CPU/GPU or the RAM. Therefore, the coefficient ω may vary depending on the platform. Usually, a pre-experiment on a small part of the dataset can help us to evaluate this coefficient for a specific classifier. Combining these two parts, the definition of FPTC is as follows:

$$t^* = F(n, m, v, \boldsymbol{\theta}') \tag{1}$$

$$t' = \omega \times t^* \tag{2}$$

where t' is the real running time, ω is the coefficient, and t^* is the time estimated by structural analysis. In the following section, we will derive the algorithm-related part of FPTC for five classically and commonly used classifiers in the remote sensing field. We derive the FPTC of the selected algorithms in the following order: *k*NN, LR, CART, RF, and SVM.

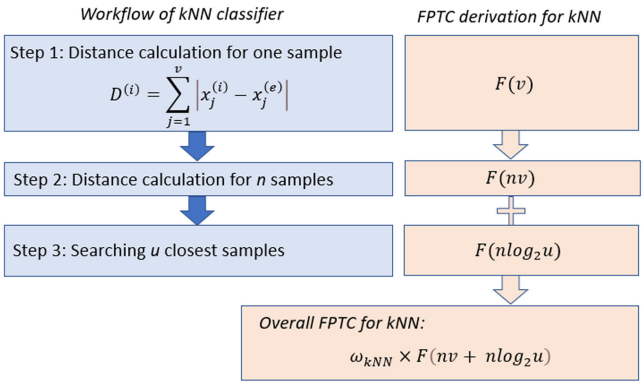


Fig. 1. Deriving FPTC for kNN.

B. Deriving FPTC for k-NN

The kNN classifier [14] memorizes the entire training data $\boldsymbol{T}_r$ and performs classification only if the test data $\boldsymbol{x}^{(e)} \in \boldsymbol{T}_e$ are given, in which $e \in \{n+1, n+2, \ldots, n+b\}$. The classifiers compute the distance or similarity between the test data $\boldsymbol{x}^{(e)}$ and each training sample. The classifier then found a group of u objects in the training set $\boldsymbol{T}_r$ that were closest to the test object $\boldsymbol{x}^{(e)}$. In this process, the commonly used Manhattan distance is equivalent to the Minkowskian r-distance function with $r = 1$, adopted as the distance or similarity metric with the following [19]:

$$D^{(i)} = \sum_{j+1}^{v} \left| x_j^{(i)} - x_j^{(e)} \right| \tag{3}$$

where $D^{(i)}$ is the distance between the training sample $(\boldsymbol{x}^{(i)}, y^{(i)})$ and the test data $\boldsymbol{T}_e$, in which $i \in \{1, 2, 3, \ldots\ldots, n\}$.

The kNN classifier is a lazy learner, which means that the cost of building the model is cheap, but classifying the test samples is relatively expensive [20]. To calculate the complexity over testing samples, the FPTC of kNN classifiers is divided into two parts (see Fig. 1). First, the FPTC of calculating the distance between one training sample and test data $\boldsymbol{x}^{(e)}$ is $F(v)$ (3). When there are n testing samples that need to be classified, the FPTC becomes $F(vn)$. Second, training samples with a minimum distance should be selected from the training set. In this classic optimal searching algorithm, the FPTC becomes $F(n\log_2 u)$. Therefore, the total FPTC of kNN is $F(nv + n\log_2 u)$. Considering the corresponding coefficient $\omega_{k\text{NN}}$, the FPTC of kNN is associated with the real running time $t'_{k\text{NN}}$ as follows:

$$t'_{k\text{NN}} = \omega_{k\text{NN}} \times t^*_{k\text{NN}} = \omega_{k\text{NN}} \times F(nv + n\log_2 u). \tag{4}$$

C. Deriving FPTC for LR

LR [15] performs multiple classifications by replacing the posterior probabilities of sigmoid transformation with that of softmax transformation [21]. In our derivation, the L2 norm is added into the loss function as a regularization term, which improves the numerical stability and robustness of the LR classifier. According to our derivation, the loss function of LR with the L2 norm and softmax takes the following form [21]:

$$J(\boldsymbol{\theta}) = -\frac{1}{n} \cdot \sum_{i=1}^{n} \sum_{j=1}^{m} 1\left\{ y^{(i)} = j \right\} \cdot \log \frac{\exp(\boldsymbol{\theta}_j^T \boldsymbol{x}^{(i)})}{\sum_{l=1}^{m} \exp(\boldsymbol{\theta}_l^T \boldsymbol{x}^{(i)})} + \frac{a}{2} \cdot \| \boldsymbol{\theta} \|_2^2 \tag{5}$$

where $\boldsymbol{\theta}$ is the $v \times m$ matrix, in which elements are the parameters of LR, θ_{pj} is the weight of the j category in the feature layer p, and α controls the regularization strength.

The goal of the LR classifier is to find the parameters of $\boldsymbol{\theta}$ with optimal values when the minimum value of loss function is obtained. The stochastic average gradient (SAG) is a common strategy to optimize the LR classifier [22]. The SAG is an improvement on stochastic gradient descent, and only proposed general equation in the original paper. According to our derivation, $\boldsymbol{\theta}_{\boldsymbol{j}}^{\boldsymbol{T}} = \{\theta_{1j}, \theta_{2j}, \theta_{3j}, \ldots\ldots, \theta_{vj}\}$ of LR with L2 norm and softmax transformation are updated according to [22]

$$\boldsymbol{\theta}_j^{r+1} = \boldsymbol{\theta}_j^r - \frac{\lambda}{n} \sum_{i=1}^{n} z_i^r \tag{6}$$

$$\boldsymbol{z}_i^r = \begin{cases} \nabla_{\boldsymbol{\theta}_j} J(\boldsymbol{\theta}) & \text{if } i = i_r \\ z_i^r & \text{otherwise} \end{cases} \tag{7}$$

$$\nabla_{\boldsymbol{\theta}_j} J(\boldsymbol{\theta}) = \boldsymbol{x}^{(i)} \left(1\left\{ y^{(i)} = j \right\} \right) - \frac{\exp(\boldsymbol{\theta}_j^T \boldsymbol{x}^{(i)})}{\sum_{l=1}^{m} \exp(\boldsymbol{\theta}_l^T \boldsymbol{x}^{(i)})} + \alpha \boldsymbol{\theta}_j \tag{8}$$

where λ is the learning rate, and i_r is taken at random from the set $\{1, 2, 3, \ldots, n\}$ for the rth iteration.

For each iteration, (6)–(8) are updated once. Therefore, the loss function is calculated one at a time by (8), and $\boldsymbol{\theta}_{\boldsymbol{j}}$ with v elements is updated one at a time by (7). Based on the above-mentioned analysis, the FPTC of each iteration is $F(mvn)$. In multiple LR classification, supposing that Q iterations (the number of iterations during the SAG process) are carried out for each category, the total FPTC of LR becomes $F(Qmvn)$. In this case, the FPTC of multiple LR in the m category can also be written as $F(Qm^2vn)$. Finally, the FPTC of LR is associated with a real running time t'_{LR} as follows, and the key steps for derivating FPTC for LR are shown in Fig. 2

$$t'_{\text{LR}} = \omega_{\text{LR}} \times t^*_{\text{LR}} = \omega_{\text{LR}} \times F(Qm^2vn). \tag{9}$$

D. Deriving FPTC for CART

CART focuses on compiling training data by the Gini splitting rule [16], [17]. The Gini index is similar to the entropy or information-gain criterion for constructing the $IF(\boldsymbol{T}_r)$ impurity function [20]. The $IF(\boldsymbol{T}_r)$ impurity is used to determine the split nodes of the binary tree and takes the following form [16]:

$$IF(\boldsymbol{T}_r) = 1 - \sum_{j=1}^{m} p(j/T_r)^2 = 1 - \sum_{j=1}^{m} \left(\frac{\sum_i^n 1\left\{ y^{(i)} = j \right\}}{n} \right)^2 \tag{10}$$

where $p(j/T_r)$ is the ratio of the number of samples in class j to the total number of samples in training data $\boldsymbol{T}_r$.

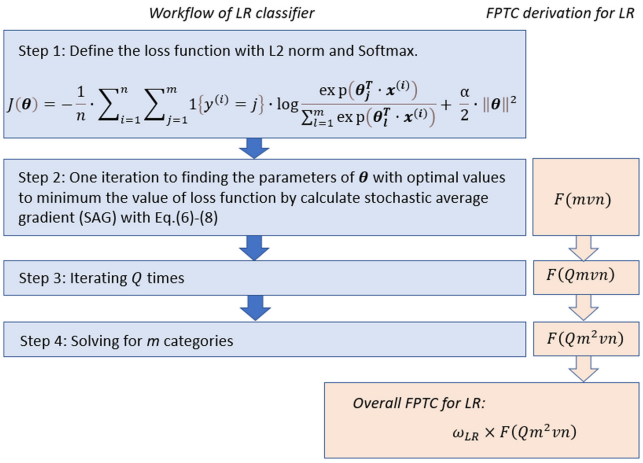


Fig. 2. Deriving FPTC for LR.

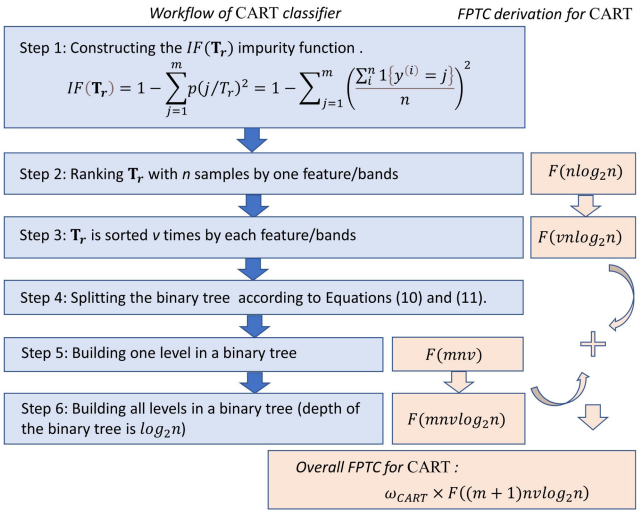


Fig. 3. Deriving FPTC for CART.

Suppose c is the split node for feature/band q. The binary tree is split into two subtrees $\boldsymbol{T}_l$ and $\boldsymbol{T}_g$, in which $\boldsymbol{T}_l=\{(\boldsymbol{x},y)|x_q<c\}$ and $\boldsymbol{T}_g=\{(\boldsymbol{x},y)|x_q>c\}$. The goal of the CART classifier is to find the optimal c^* and q^*, which minimize the sum of $IF(\boldsymbol{T}_l)$ and $IF(\boldsymbol{T}_g)$. The sum of the $IF(\boldsymbol{T}_l)$ and $IF(\boldsymbol{T}_g)$ is the optimization function. To make it easier to understand, we have made a simple change to the optimization function [16]

$$J(q,c)=-\frac{n_l}{n}\sum_{j=1}^{m}p(j/T_l)^2-\frac{n_g}{n}\sum_{j=1}^{m}p(j/T_g)^2 \tag{11}$$

$$c^*,q^*=\arg\min_{c,q}J(c,q) \tag{12}$$

where n_l is the number of samples in $\boldsymbol{T}_l$, and n_g is the number of samples in $\boldsymbol{T}_g$.

In the CART classifier, FPTC is divided into two parts (see Fig. 3). First, $\boldsymbol{T}_r$ is sorted v times by each feature in advance, which only needs to be done once. Ranking n samples is a classic computer problem, and the FPTC of its optimal algorithm is $F(n\log_2 n)$. The FPTC of v times ranking is then $F(vn\log_2 n)$.

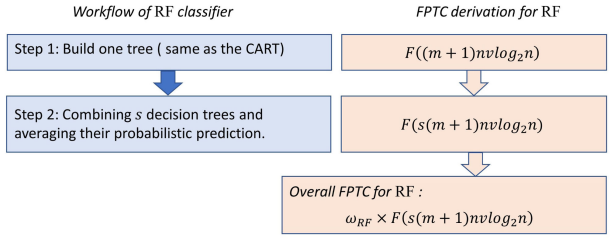


Fig. 4. Deriving FPTC for RF.

Second, the binary tree is split according to (11) and (12). The FPTC of this process is $F(mnv)$. A binary decision tree is established when the training data are split repeatedly. According to previous studies, the average expected tree depth is known as $\log_2 n$ [9]. Building every level in this binary tree costs $F(mnv)$. Thus, the FPTC of this part is $F(mnv\log_2 n)$. By adding the FPTC of the two parts, the FPTC of CART is $F((m+1)nv\log_2 n)$. In order to make the FPTC less redundant and more concise, when m is large enough, it can be simplified to $F(mnv\log_2 n)$ with $\frac{1}{m+1}$ error rate. The FPTC of CART is associated with the real running time t'_{CART} as follows, and the key steps for derivating FPTC for RF are shown in Fig. 3:

$$t'_{\text{CART}}=\omega_{\text{CART}}\times t^*_{\text{CART}}=\omega_{\text{CART}}\times F((m+1)nv\log_2 n). \tag{13}$$

E. Deriving FPTC for RF

RF is a type of ensemble learning [17]. The goal of the ensemble method is to combine several weak learners in order to improve the performance of the classifier. The ensemble methods used in RF include bagging, boosting, bootstrap, etc. The bagging, as the best ensemble method with a strong and complex model, is used in our study. Without special requirement, the bagging method randomly extracts the n sample size subset with a replacement from the training set. Generally, CART is often used as a weak learner in RF. When we build an RF with an s tree, s subsets are extracted and used to build an s CART.

When we build a CART tree with n samples, the average expected tree depth is $\log_2 n$. Every level cost is F(mnv), and the FPTC of this part is F($mnv\log_2 n$) [9]. When adding the sorting time F($vn\log_2 n$), the FPTC of building one tree is $F((m+1)nv\log_2 n)$, so the FPTC of RF with s trees is $F(s(m+1)nv\log_2 n)$. Similarly, the FPTC of RF can be simplified to $F(smnv\log_2 n)$ with a $\frac{1}{m+1}$ error rate when m is large enough. The FPTC of RF is associated with the real running time t'_{RF} as follows, and the key steps for derivating FPTC for RF are shown in Fig. 4:

$$t'_{\text{RF}}=\omega_{\text{RF}}\times t^*_{\text{RF}}=\omega_{\text{RF}}\times F(s(m+1)nv\log_2 n). \tag{14}$$

F. Deriving FPTC for SVM

SVM [18] aims to find the hyperplane with the maximum distance from the support vectors, which is often treated as a linear programming problem to transform it into a dual problem by the Lagrangian multiplier method [23], [24]. The function of

hyperplane $g(\boldsymbol{x}, \boldsymbol{\theta})$ takes the following form [18]:

$$g(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x} + b = \sum_{j=1}^{v} \theta_v x_j + b. \tag{15}$$

In general, a soft margin hyperplane [24] is often used to avoid overfitting, which is done by introducing the variable ξ. Based on the Lagrangian multiplier method, the loss function with a soft margin takes the following form [18], [24]:

$$\begin{aligned} L(\boldsymbol{\theta}, b, \boldsymbol{a}, \boldsymbol{\xi}, \boldsymbol{\mu}) = \frac{1}{2} \parallel \boldsymbol{\theta} \parallel^2 + C\sum_{i=1}^{n} \xi_i \\ + \sum_{i=1}^{n} \alpha_i (1 - \xi_i - y^{(i)}(\boldsymbol{\theta}^{\boldsymbol{T}}\boldsymbol{x}^{(\boldsymbol{i})} + \boldsymbol{b})) \\ - \sum_{i=1}^{n} \mu_i \xi_i \end{aligned} \tag{16}$$

where C is the regularization parameter and $\alpha_i \geq 0, \mu_i \geq 0$. The dual question of finding the hyperplane with the maximum distance in SVM with soft margin is as follows [18], [24]:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_i y^{(i)} y^{(j)} K_{ij} \tag{17}$$

$$\text{s.t.} \sum_{i=1}^{n} \alpha_i y^{(i)} = 0, \alpha_i \geq 0, i = 1, 2, \ldots\ldots, n \tag{18}$$

where $K_{i,j} = K(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp(-\gamma \times \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)(\gamma > 0)$ is the radial basis function (RBF). In our study, the RBF, as the most commonly kernel function, is used in our research. The kernel choice can be a key issue for the SVM performance, since the different kernel could lead to different results. From the perspective of time complexity, they are the same. For instance, the time complexity of RBF is $F(v)$ (where v represents the number of the features of inputs), which is identical to the time complexity of the linear kernel. From this perspective, the "kernel trick" is efficient by measuring the difference between samples in the higher dimensional space without actually projecting them into it. Based on this reason, the different kernel function selection, such as linear, polynomial, sigmoid, or RBF, will not have a significant effect on the FPTC in SVM.

Karush–Kuhn–Tucker (KKT) conditions are as follows [18], [24]:

$$\alpha_i \geq 0, \mu_i \geq 0 \tag{19}$$

$$y^{(i)} \cdot g(\boldsymbol{x}^{(i)}) - 1 + \xi_i \geq 0 \tag{20}$$

$$\alpha_i \cdot (y^{(i)} \cdot g(\boldsymbol{x}^{(i)}) - 1 + \xi_i) = 0 \tag{21}$$

$$\xi_i \geq 0, \mu_i \xi_i = 0. \tag{22}$$

It is difficult and costly to solve the dual problem directly. In our study, sequential minimal optimization (SMO) [25] is considered a decomposition method to conquer this difficulty. At each iteration, we solve a simple two-variable problem (α_i and α_j) without needing any optimization software [25]

$$\alpha_j^{r+1} = \alpha_j^r + \frac{y^{(j)}((g(\boldsymbol{x}^{(i)}) - y^i) - (g(\boldsymbol{x}^{(j)}) - y^j))}{K_{i,i} + K_{j,j} - 2K_{i,j}} \tag{23}$$

$$\alpha_i^{r+1} = \alpha_i^r + y_i y_j (a_j^r - a_j^{r+1}) \tag{24}$$

when α_i^{r+1} is not at the bounds, b is calculated as follows [25]:

$$\begin{aligned} b^{r+1} = g(\boldsymbol{x}^{(i)}) - y^{(i)} + y_i(\alpha_i^{r+1} - \alpha_i^r)K_{i,i} \\ + y_j(\alpha_j^{(r+1)} - \alpha_j^r)K_{i,j} + b^r \end{aligned} \tag{25}$$

when α_j^{r+1} is not at the bounds, b is calculated as follows [25]:

$$\begin{aligned} b^{r+1} = g(\boldsymbol{x}^{(j)}) - y^{(j)} + y_i(\alpha_i^{r+1} - \alpha_i^r)K_{i,j} \\ + y_j(\alpha_j^{(r+1)} - \alpha_j^r)K_{j,j} + b^r \end{aligned} \tag{26}$$

when neither α_i^{r+1} nor α_j^{r+1} is at bounds, (25) and (26) are equal. When both α_i^{r+1} and α_j^{r+1} are at bounds, b^{r+1} is halfway between (25) and (26). The k is the number of support vectors and $\boldsymbol{x}^{(l)} \in \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots\ldots, \boldsymbol{x}^{(k)}\}$ is the support vector. Notably, the SMO optimizer [26] has been replaced with the SMO-type optimizer since LIBSVM V2.8. The SMO-type optimizer may speed up convergence and reduce the number of iterations. When we consider the FPTC in one iteration, the difference in times of the FPTC between SMO and the SMO-type optimizer can be neglected.

In each iteration, all training samples are calculated and judged whether they are in KKT conditions (19)–(22). We selected the one that does not comply with KKT $(x^{(i)}, y^{(j)})$ (21) and the one $(x^{(i)}, y^{(j)})$ with the greatest distance from it. We then solved α_i, α_j, θ, and b. The FPTC of each iteration is $F(nv + vk)$. KKT conditions are updated with every iteration, and the loop is stopped when all training samples fit the KKT conditions or we reach the maximum number of iterations we set. Thus, the FPTC of iteration Q (the number of iterations during the SMO process) is $F(Qv(n + k))$.

Originally, SVM was designed for binary classification, and it is not good at multiple classifications. There are still several multiclassification methods for SVM; one-versus-one (OVO) or pairwise is one of them. In OVO, an SVM classifier needs to be built between each of the two classes, i.e., $m(m-1)/2$ classifiers need to be built for m categories. Thus, the FPTC of SVM for multiclassification is $F(m^2Qv\ (n + k))$. It is associated with the real running time t'_{SVM} as follows, and the key steps for derivating FPTC for SVM are shown in Fig. 5:

$$t'_{\text{SVM}} = \omega_{\text{SVM}} \times t^*_{\text{SVM}} = \omega_{\text{SVM}} \times F(m^2Qv(n + k)). \tag{27}$$

III. Study Area and Datasets

To test the accuracy of the FPTC derived previously, one study area and the remote sensing data were selected.

A. Study Area

The study area is located in the middle of the Xinjiang Uygur Autonomous Region, China (42°7′–42°13′ N, 86°13′–86°22′ E) (see Fig. 6). This area covers an extensive area of around

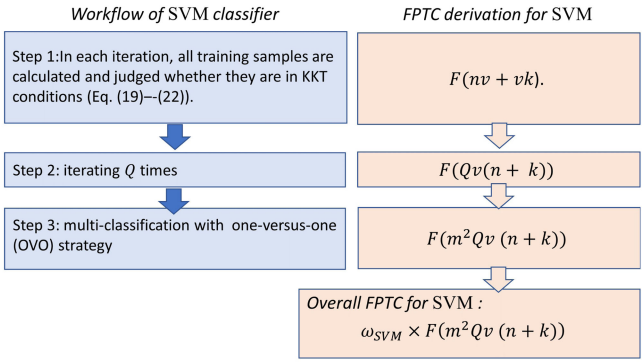


Fig. 5. Deriving FPTC for SVM.

TABLE I
ALGORITHM PARAMETER SET UP AND SOURCE OF CODES

Classifier	Number of groups	Sample size
kNN LR CART RF	29	1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10,000, 20,000, 30,000, 40,000, 50,000, 60,000, 70,000, 80,000, 90,000, 100,000, 110,000, 120,000, 130,000, 140,000, 150,000, 160,000, 170,000 180,000, 190,000, 200,000
SVM	10	1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10,000

1 660 000 km^2, mainly covered by grassland and sandy desert with a typical continental climate. Forest areas are sparsely scattered within the high mountains and along the rivers. Oasis landscapes have developed within inland river deltas, alluvial–diluvial plains, and along the edges of diluvial–alluvial fans. Agricultural land and human settlements are distributed around these areas.

With the rapid growth of the population in recent years, the ecosystem in this area faces a great challenge in terms of the dramatic change of land cover combined with changing precipitation patterns [27]. The frequency of the occurrence of natural disasters, such as sandstorms, floods, and snowstorms, increases as the global climate changes. Under these circumstances, a rapid assessment of land cover change after a disaster not only limits the loss of life and property but also provides data for emergency managers to optimize the emergency response procedure.

B. Datasets

Sentinel-2 provides continuous high-resolution images with a multispectral instrument. With the high revisit frequency (five days combined with Sentinel 2 A and 2B), Sentinel-2 imagery has been widely used for land cover mapping, change detection, and emergency response [28]. Compared with other public and free multispectral products, Sentinel-2 contains bands covering the red edge [29], which can provide indispensable information for land-cover mapping, land change detection, and the retrieval of other geophysical variables [28], [29]. In our study, a Sentinel-2 A image acquired on 8 September 2016 was downloaded from the United States Geological Survey.[1] This image corresponds to Level-1 C products, which are radiometrically and geometrically corrected top-of-atmosphere products with subpixel multispectral registration [29]. This image is cloud free, so the atmospheric correction procedure has little influence on classification in this article [30], [31].

We selected an area (2048 × 2048 pixels) from the original Sentinel-2 A image [see Fig. 6(c)]. The spatial resolutions of B5, B6, B7, B8a, B11, and B12 were resampled to 10 m by nearest-neighbor resampling [29].

Based on the field investigation, the land cover classification system in the study area was established. Eight typical land cover types, farmland, orchard, forest, grassland, water, residential area, roadway, and idle land (see Fig. 7) were selected as land cover types at the level-1 category, which remains the same as the study from other groups in this area, such as Gong and Howarth [32] and Gong *et al.* [33]. The image was manually interpreted to create a digital land cover map through image interpretation with intensive field samples (done in October 2016) over this area [see Fig. 7(b)]. Both imageries with 12 bands and the land cover map provide sufficient training samples for land cover classification in this area. To understand how time consumption changes under the different bands or the different training sample size, the training sample is divided into different groups, with samples randomly selected over the land cover map. Considering the slow training speed of SVM, only nine training groups for 1000 to 10 000 samples were prepared for SVM. Table I shows the details of the subtraining sample settings.

All classifiers are programmed by Python 3.6 (Python Software Foundation. Python Language Reference, version 3.6).[2] The SVM classifier is programmed using LIBSVM [34], while the rest of the classifiers are programmed using Scikit-learn [35]. All experiments ran on the Ubuntu 14.0[3] platform, which was equipped with an Intel Xeon e5-2620 CPU and four TITAN XP graphics cards.

C. Assessment

To verify the accuracy of the FPTC, three assessments were applied: 1) a 1:1 plot was used to compare the real running time to the FPTC, 2) we estimated the running time by FPTC and calculated the root-mean-squared error (RMSE) between the estimated and the observed running time, and 3) we compared the FPTC and the TTC with respect to the real observed running time under different feature selections.

IV. RESULTS

A. Validation of the FPTC and Correction Coefficient

We selected subtraining samples randomly from the training dataset and constructed sets of subtraining samples (see Table I). These samples were classified, and the real running time was recorded. As we can see from Fig. 8, the linear relationship

[1][Online]. Available: https://earthexplorer.usgs.gov/

[2][Online]. Available: https://www.python.org/

[3][Online]. Available: https://ubuntu.com/

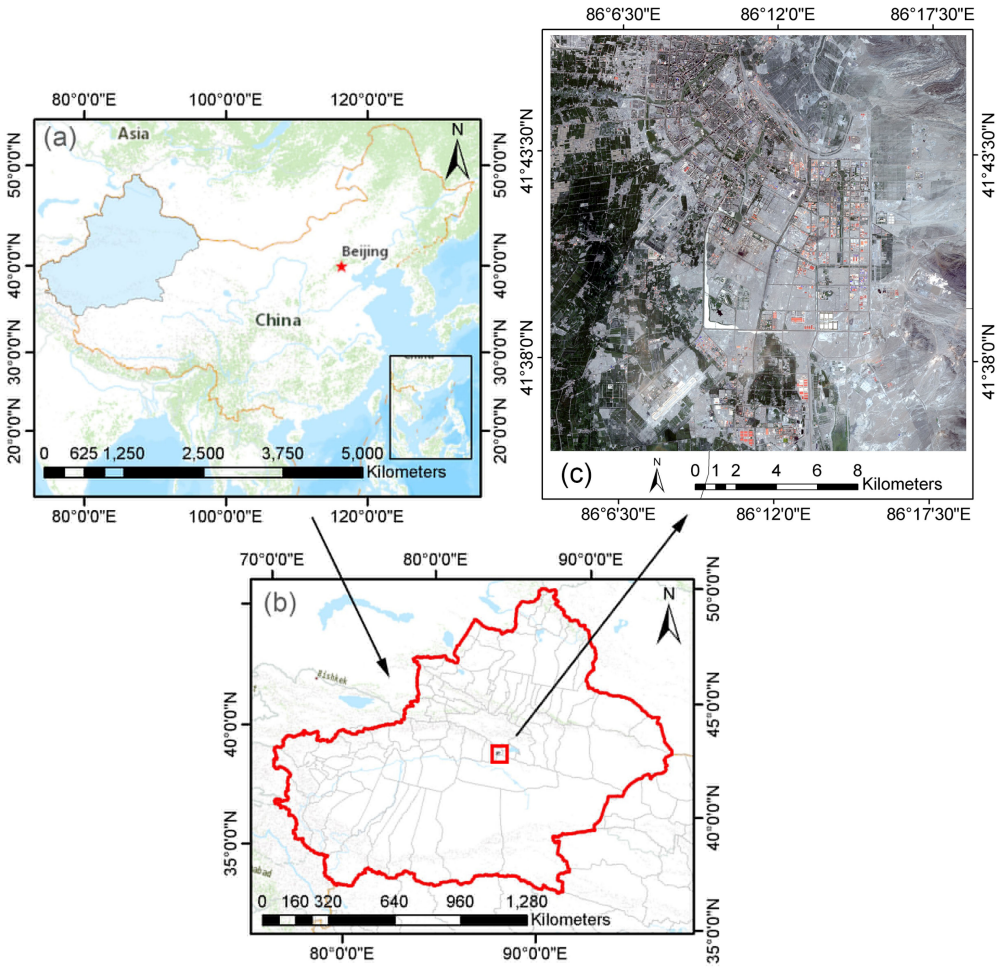


Fig. 6. Study area: (a) China, (b) Xinjiang Uygur Autonomous Region, and (c) location of the study area.

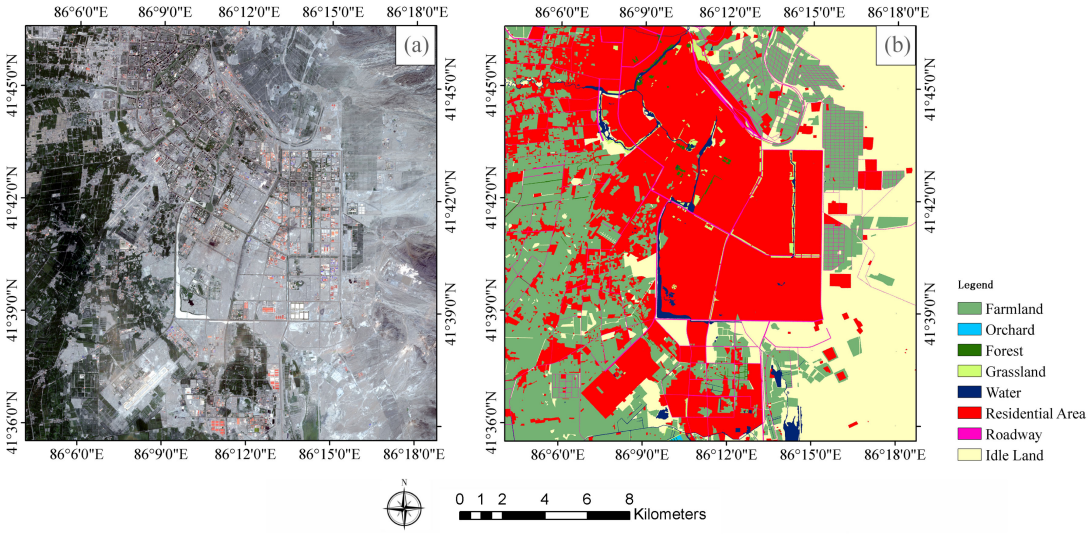


Fig. 7. Location of the selected image: (a) image; and (b) ground truth.

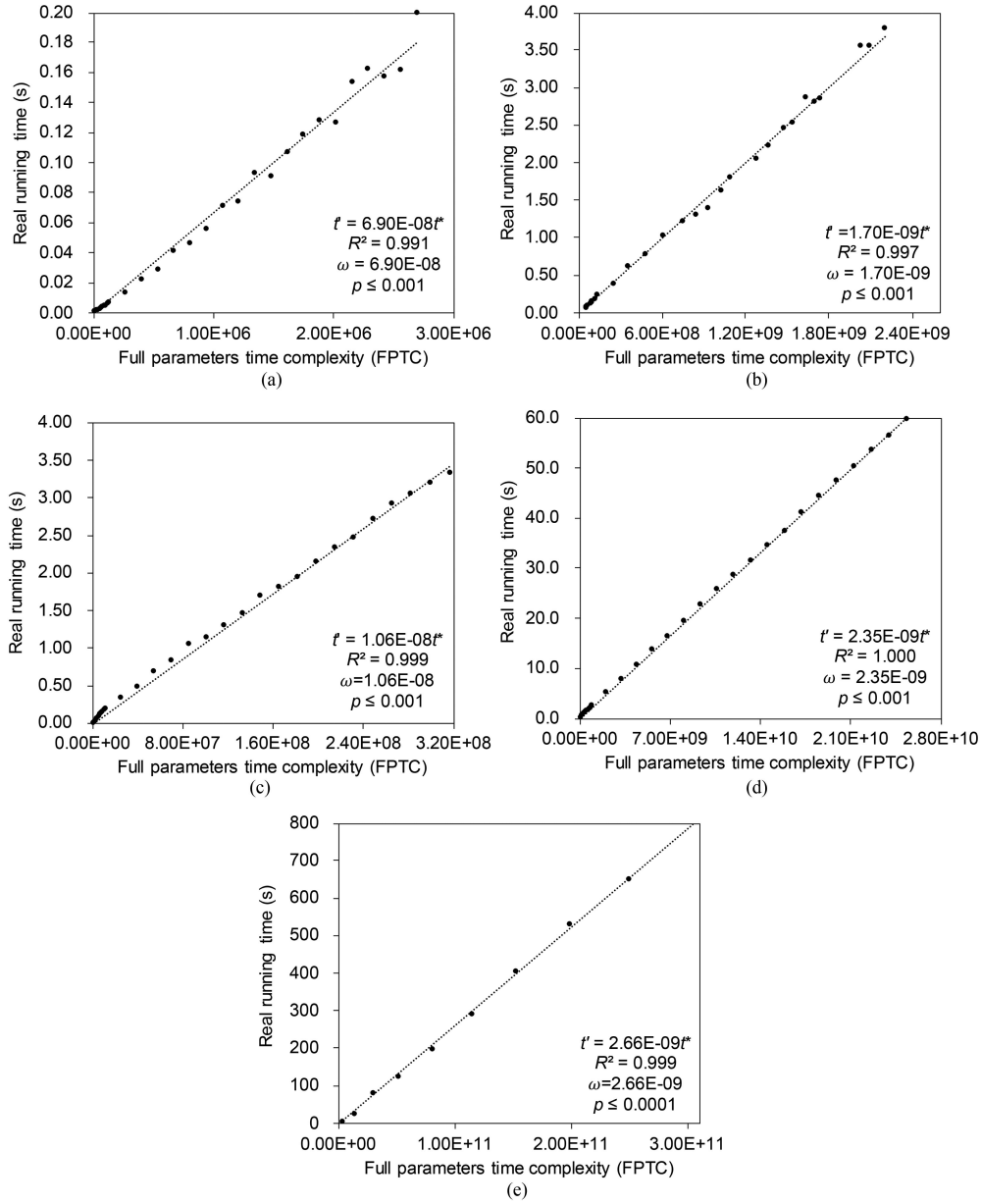


Fig. 8. Relationship between FPTC and the real running time.

between the FPTC and the real running time indicates the effectiveness of FPTC. The R-squared values for the 1:1 plot of the five classifiers were all larger than 0.99 (kNN: 0.991, LR: 0.997, CART: 0.999, RF: 1.000, and SVM: 0.999), which indicates that the linear relationship between the algorithm part of the FPTC, and the real running time is extremely strong ($p < 0.001$).

The coefficient ω of each algorithm can be found from the slope of the regression line. For *k*NN, the correction coefficient $\omega_{k\text{NN}} = 6.90E-8$, which is obtained from the linear relationship ($t'_{k\text{NN}} = 6.90E-8t^*_{k\text{NN}},\ R^2 = 0.991,\ p \leq 0.001$). This relationship shows that the actual running time (in seconds) is equal to FPTC $_{k\text{NN}} \times \omega_{k\text{NN}}$ [see Fig. 8(a)]. Similarly, for LR, the coefficient $\omega_{\text{LR}} = 1.70E-9$ has a linear relationship ($t'_{\text{LR}} = 1.70E-09t^*_{\text{LR}},\ R^2 = 0.997,\ p \leq 0.001$) [see Fig. 8(b)]. For CART, the correction coefficient $\omega_{\text{CART}} = 1.06E-08$, which has a linear relationship ($t'_{\text{CART}} = 1.06E-08t^*_{\text{CART}},\ R^2 = 0.999,\ p \leq 0.001$) [see Fig. 8(c)]. For RF, the correction coefficient $\omega_{\text{RF}} = 2.35E-9$, which has a linear relationship ($t'_{\text{RF}} = 2.35E-9t^*_{\text{RF}},\ R^2 = 1.000,\ p \leq 0.001$) [see Fig. 8(d)]. For SVM, the correction coefficient $\omega_{\text{SVM}} = 2.66E-9$, which has a linear relationship ($t'_{\text{SVM}} = 2.66E-9t^*_{\text{SVM}},\ R^2 = 0.999,\ p \leq 0.001$) [see Fig. 8(e)].

The slope can be roughly estimated based on the two available datasets, regardless of the magnitude of the training data. This means that this value can be obtained by prerunning the algorithm under two small parts of the total dataset. As a result that the coefficient ω represents the physical part of the FPTC, this value only varies when the algorithm is applied to different computational environments.

TABLE II
ESTIMATED AND REAL RUNNING TIMES FOR EACH ALGORITHM

Classifier	Parameter choice	FPTC	ω	Average estimated running time (s)	Real running time (s)	RMSE (s)
kNN	$v = 10; n = 100,000; k = 11$	$1.35E+06$	$6.90E-08$	0.093	0.081	0.01
LR	$v = 10; n = 100,000; C = 1E-05; Q = 17$	$1.09E+09$	$1.70E-09$	1.850	1.741	0.18
CART	$v = 10; n = 100,000;$	$1.49E+08$	$1.06E-08$	1.585	1.632	0.16
RF	$v = 10; n = 100,000; s = 80$	$1.20E+10$	$2.35E-09$	28.104	28.445	0.57
SVM	$v = 10; n = 100,000; C = 100;$ γ = 8; Q = 44355; k = 7290	$3.05E+11$	$2.66E-09$	810.055	803.800	15.76

TABLE III
COMPARISON OF TTC AND FPTC

Classifier	ω	FPTC	TTC	Real running time (s) (100,000 samples)
kNN	$6.90E-08$	$F(nv + nlog_2u)$	$O(n)$	0.081
LR	$1.70E-09$	$F(m^2vn)$	$O(n)$	1.741
CART	$1.06E-08$	$F((m+1)nvlog_2n)$	$O(nlog_2n)$	1.632
RF	$2.35E-09$	$F(s(m+1)nvlog_2n)$	$O(nlog_2n)$	28.445
SVM	$2.66E-09$	$F(m^2vn^2)$	$O(n^2)$	803.800

B. Estimating the Real Running Time With the FPTC

Based on the strong linear relationship provided by Fig. 8, the running time can be estimated accurately. To calculate the RMSE between the estimated and observed running time, we fixed samples at 100 000 and recorded the real running time. Table II shows the estimated and actual running times for each algorithm.

Regarding the FPTC of each classifier, the highest FPTC comes from SVM (3.05E+11), followed by RF (1.20E+10), LR (1.09E+09), and CART (1.49E+08); the lowest FPTC, produced by kNN, was 1.35E+06 (shown in Table II).

Table II shows that the real running time keeps a certain consistency with the estimated running time. From Table II, we can see that the highest real running time was achieved by SVM (803.800 s), while the estimated running time of SVM was 810.055 s. The next highest real running time was achieved by RF (28.445 s), while the estimated running time of RF was 28.104 s. LR (1.741 s), CART (1.632 s), and kNN (0.081 s) produced the lowest real running times, while the estimated running times of LR, CART, and kNN were 1.850 s, 1.585 s, and 0.093 s, respectively (see Table II). Regarding the RMSE of each classifier, the highest RMSE came from SVM (15.76 s), followed by RF (0.57 s), LR (0.18 s), CART (0.16 s), and kNN (0.01 s) (shown in Table II). The average RMSE between the real running time and the estimated running time was 3.34 s, which shows that the real running time can be estimated accurately by FPTC.

C. Comparing FPTC and TTC

How does the performance of FPTC compare to TTC? In this section, we compare FPTC to TTC using both theoretical and experimental methods.

Firstly, at a theoretical level, TTC has a limited ability to show the difference in running time between the different algorithms. For instance, the TTC of kNN and LR is $O(n)$, while the TTC of CART and RF is $O(n\log_2 n)$ (see Table III). If we use TTC for the evaluation of running time, there is no difference between the running time of kNN and LR or between CART and RF. From our experiments with 100 000 samples, the real running time of kNN was 0.081 s, and the real running time of LR was 1.741 s. Similarly, a noticeable difference exists between CART and RF. The real running time of CART was 1.632 s, while the real running time of RF was 28.445 s (see Table III).

The TTC of LR is $O(n)$, while the TTC of CART is $O(n\log_2 n)$. In terms of TTC, the running time of CART should be higher than that of LR. The reality is just the opposite. The real running time of CART was 1.632 s, which is lower than that of LR (1.741 s). The reason is that, in TTC, many low-order details and key parameters have been ignored. These details and parameters may be negligible when comparing algorithm time complexity at a theoretical level, but they are essential to estimating running time at the operational level.

The FPTC we proposed can make up for the above shortcomings of TTC. FPTC is derived by examining the overall structure of the classifier program and its mathematical principles. As we can see from Table III, the FPTC of the kNN is different from the FPTC of the LR.

In addition, FPTC can also reflect changes in running time with different parameters. When n training samples changed from low to high and other influencing parameters were kept the same, the FPTC of SVM was most vulnerable to the effects of the change, followed by RF, CART, kNN, and LR. If n changed from 1 to 128, the FPTC of LR increased 128-fold, kNN a little more than 128-fold, CART and RF 896-fold, and SVM more than 16 384-fold. When the number of categorizations m changed from low to high and other influencing parameters were kept the same, the FPTC of SVM and LR was most vulnerable to the effects of the change, followed by CART and RF; kNN was unaffected. For example, if m changed from two classes to 200 classes, the FPTC of SVM and LR increased 10 000-fold, and that of CART and RF increased 100-fold. When the number of features or bands v changed from low to high and other influencing parameters were kept the

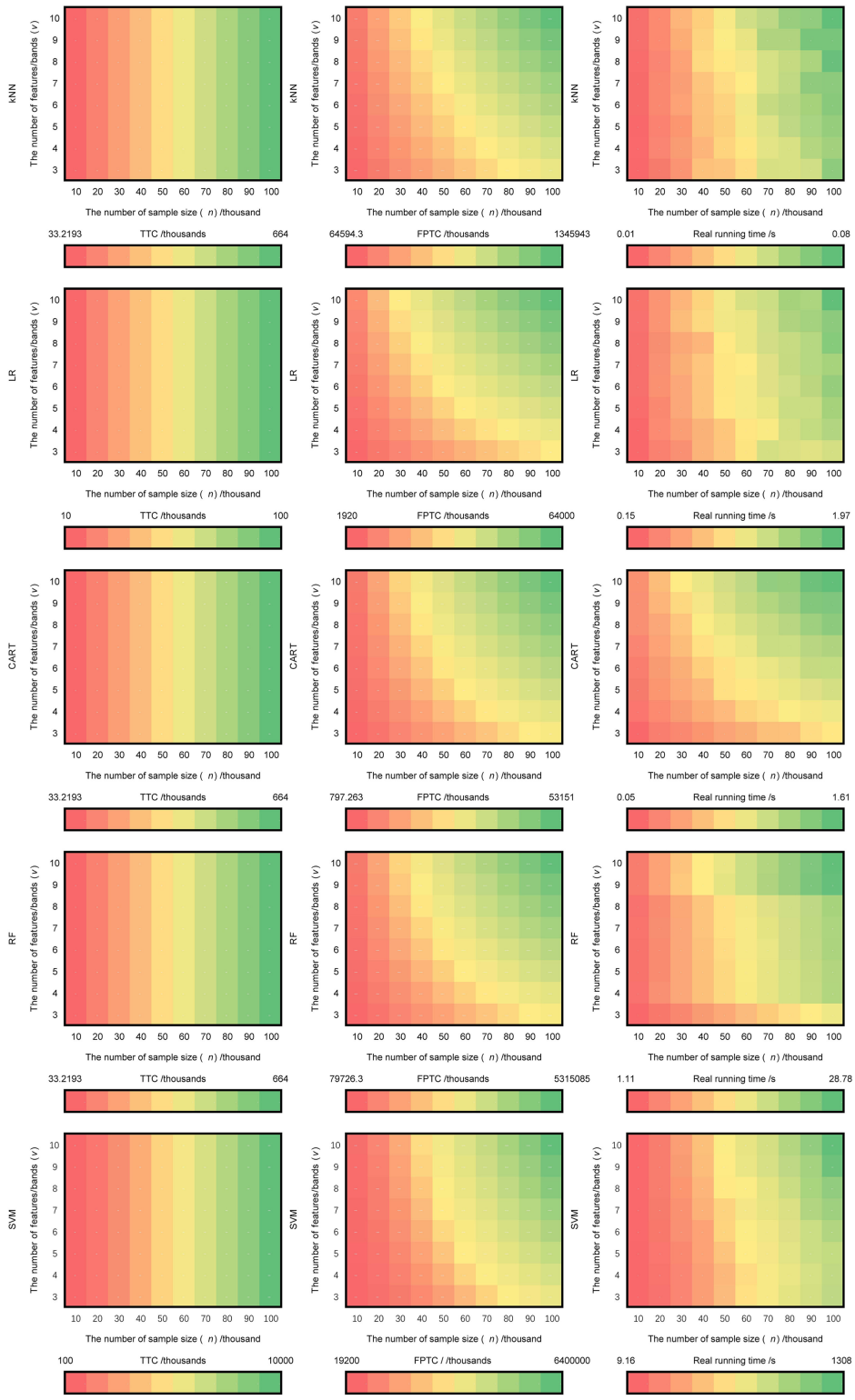


Fig. 9. Comparison between TTC (left column) and FPTC (middle column) to the real running time (right column).

same, the time complexities of SVM, LR, CART, and RF were changed within a polynomial time, while that of kNN was less affected.

Secondly, to further illustrate the difference between FPTC and TTC from experiments, we analyzed the changing trends of FPTC and TTC under all combinations of different bands ($v = 3, 4, 5, \ldots, 10$) and different sample sizes ($n = 10, 20, 30, \ldots, 100\,000$) and compared them with real running time trends. The values of TTC, FPTC, and the real running times are mapped from low to high in red to green in Fig. 9. The results show that TTC does not respond to changes in v, while FPTC can better reflect the variation of v.

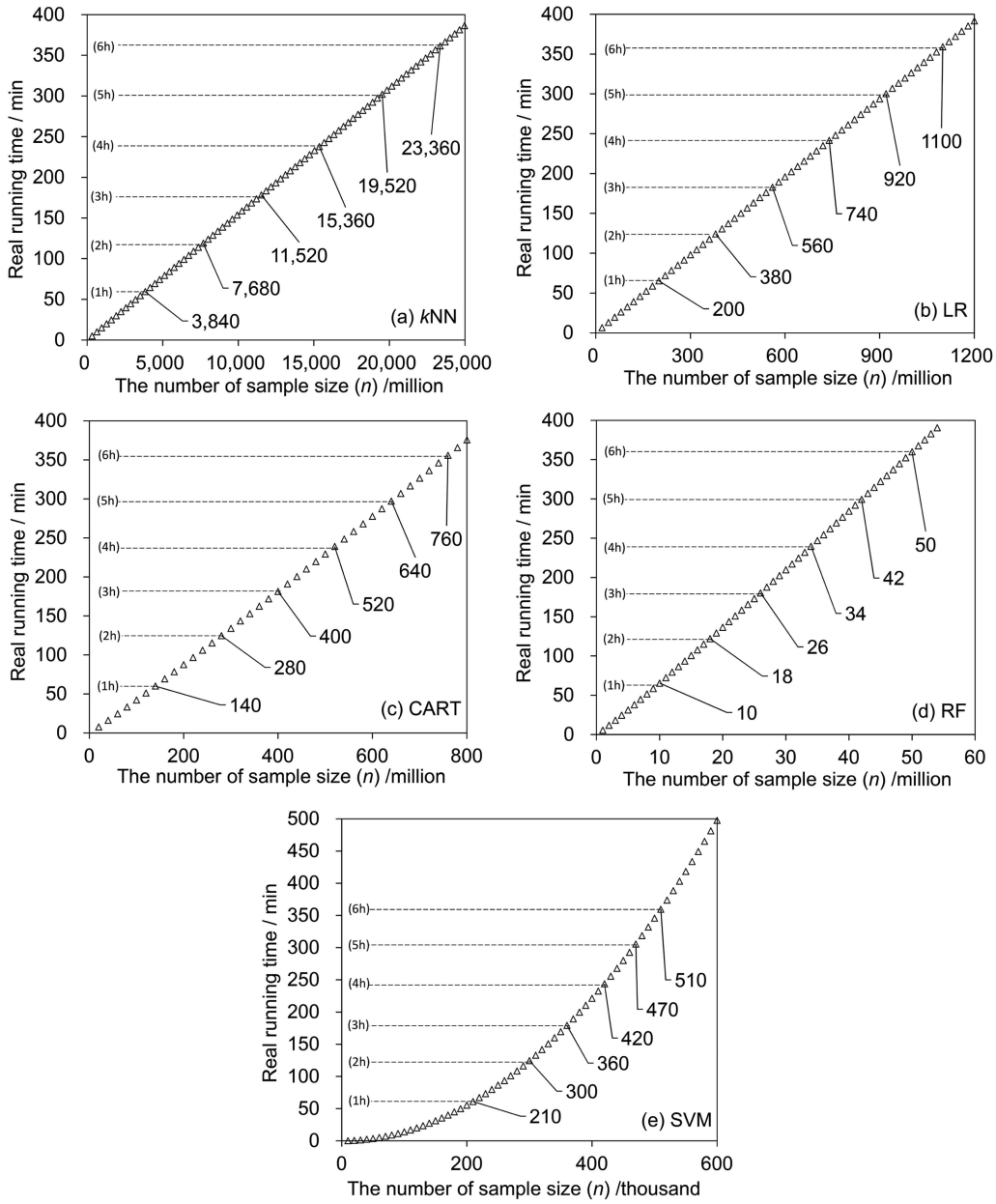


Fig. 10. Capacity of each algorithm.

As we can see, FPTC shows a similar pattern to the real running time under different bands and data sizes. The pattern of the TTC is different since the impact of the different features/bands is ignored by TTC.

D. Simple Application of FPTC

Without estimating the running time accurately, choosing a classifier for a time-limited LULC classification task could be blind and subjective. In this section, we create an urgent classification task that should be done within 6 h. The natural question is of how many training samples we should prepare for different classifiers to fit this time limit.

Through FPTC and ω, we can estimate the maximum sample size (MSS) that can be processed within the time threshold. A larger maximum training sample means that the algorithm can handle more samples, which will improve the overall classification accuracy. Fig. 10 shows the relationship between sample size and the running time calculated by FPTC with corresponding coefficient ω. The exact number of training samples for each hour is also shown in these figures.

Taking a 1-h training limit as an example, the results show that the algorithm with the smallest MSS is SVM (0.21 million) [see Fig. 10(e)], followed by RF (10 million) [see Fig. 10(d)], CART (140 million) [see Fig. 10(c)], and LR (200 million) [see Fig. 10(b)]. The algorithm with the maximum MSS is *k*NN (3.84 billion) [see Fig. 10(a)].

In our study, the main goal was to provide a quantitative measurement for emergency managers to compare and filter different classifiers under different time and resource limits.

Threshold analysis can help us to perform classifier screening. For instance, suppose we need to process a Sentinel-2 A image (5000×5000 pixels). It would require 25 million pixels to construct the model. With SVM, it is impossible to execute the current task. If we only consider the running time, kNN could be the optimal classifier.

V. Discussion

A. Effect of Training Parameters on FTPC.

The parameters that affect the running time of an algorithm can be classified into two major categories. One is a hyperparameter, which is set before starting the learning process (e.g., the number of trees s in RF). The other is a training parameter, which can only be obtained after running (e.g., the total iterations Q and the number of support vectors k in SVM). These training parameters can only be obtained after the program runs. Thus, it will be difficult to estimate the running time of these classifiers without running them. Fortunately, the characteristics of the training parameter can be estimated by other pre-obtained parameters such as the sample size. Fig. 11 shows that the total iterations Q in SVM has a linear correlation with the number of samples ($R^2 = 1.00$) [see Fig. 11(a)]. The hyperparameter k also has the same significant linear correlations ($R^2 = 1.00$) [see Fig. 11(b)]. According to these linear correlations, parameters Q and k can be removed from $F(m^2Qv\ (n+k))$ in SVM. The real running time t'_{SVM} in (27) can then be rewritten as (28), where $\omega_{\text{SVM}}{}'$ is the new coefficient. The total FPTC of SVM is $F(m^2vn^2)$. After this, the estimated running time is no longer influenced by the training parameters.

$$\begin{aligned} t'_{\text{SVM}} &= \omega_{\text{SVM}} \times t^*_{\text{SVM}} = \omega_{\text{SVM}} \times F(m^2Qv(n+k)) \\ &= \omega'_{\text{SVM}} \times F(m^2vn^2). \end{aligned} \tag{28}$$

Another classifier with training parameters is LR. The total iterations Q in LR is generated in the SAG algorithm, which can only be obtained after iteration. Fig. 11(c) shows that the total iterations Q is stable with an average of 18.02 ± 1.16; it changes as the sample size n increases [Fig. 11(c)). Therefore, in the derivation of FPTC, Q can be approximated as a constant and combined with coefficient ω. After this correction, the total FPTC of LR is corrected to $F(m^2vn)$ (29). Thus, the estimated running time is no longer influenced by the training parameters

$$\begin{aligned} t'_{\text{LR}} &= \omega_{\text{LR}} \times t^*_{\text{LR}} = \omega_{\text{LR}} \times F(Qm^2vn)) \\ &= \omega'_{\text{LR}} \times F(m^2vn). \end{aligned} \tag{29}$$

B. Determining Coefficient ω With a Pre-Experiment

As mentioned before, the coefficient ω is the key to estimating the actual running time of classifiers. This parameter is more related to the physical computational environment, such as the compiling programs and computer hardware. Fig. 8 shows that the coefficient ω (the slope) will not change as the input data size increases, even when the data size is small. Based on this finding, the coefficient ω can be calculated through two small-scale pre-experiments. The coefficient ω can then be reused in other estimates with a larger sample size. Once the physical computational environment changes, the coefficient ω should be reevaluated under the new circumstances. This may limit the application of FPTC to a cloud computing environment since the physical computational environment may change during the realization process. How the coefficient ω is related to the computational resource (CPU or GPU frequency) still needs to be quantitatively evaluated in the future.

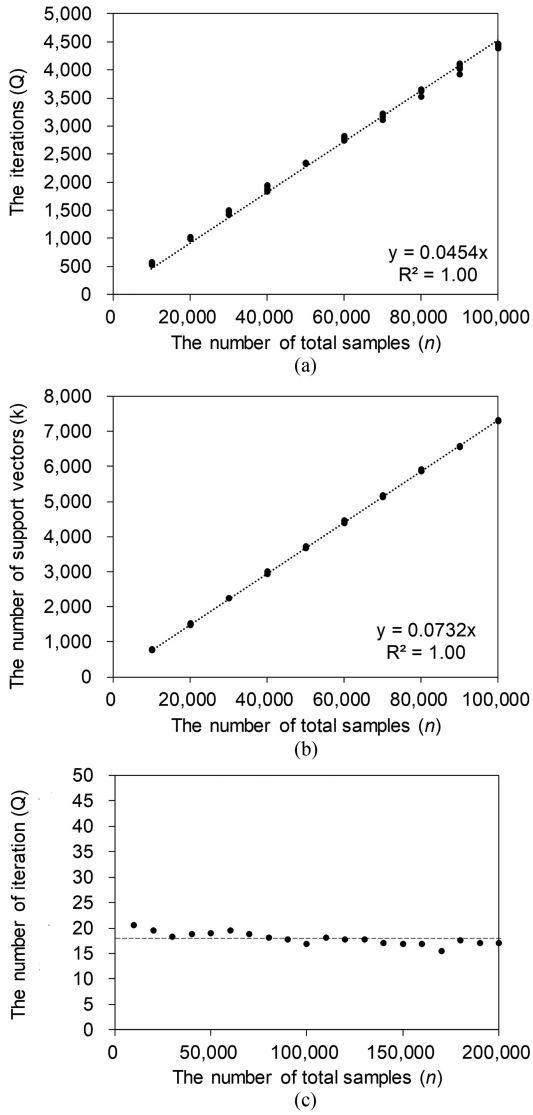


Fig. 11. Correlations between the training parameters and the number of samples.

VI. Conclusion

In emergency response to natural disasters, accurate time predictions help emergency managers to choose a classification algorithm with limited time and resources. In this article, we proposed FPTC and the coefficient ω to estimate the running time of each classifier. The FPTC of five common classifiers (kNN, LR, CART, RF, and SVM) was derived by examining the overall structure of the classifier program and its mathematical principles. A linear regression model was built based on the relationship between the real running time and the FPTC, and the coefficient ω was obtained from the linear regression. We

then accurately predicted the running time of each classifier and filtered out the appropriate classifier. The results can be summarized as follows.

1) We proposed a method to quantitatively evaluate the time efficiency of machine learning classifiers called FPTC. We derived the FPTC of five general classifiers. The results show that the FPTC of *k*NN is $F(nv + n\log_2 u)$, the FPTC of LR is $F(Qm^2vn)$, the FPTC of CART is $F((m + 1)nv\log_2 n))$, the FPTC of RF is $F(s(m + 1)nv\log_2 n)$, and the FPTC of SVM is $F(m^2Qv\ (n + k))$.
2) A strong linear relationship between the FPTC and the running time was found in our study ($R^2 \geq 0.991,\ p \leq 0.001$). This linear relationship verifies the correctness of the FPTC derivation process. The correction coefficient ω of each algorithm can be found from a strong linear regression.
3) The running time of each classifier was estimated by coefficient ω with FPTC. Our study showed that the average RMSE between the real running time and the estimated running time is 3.34 s, which shows the feasibility and accuracy of using FPTC to predict the running time of algorithms.
4) Our study showed that training parameter Q in SVM had significant linear correlations with the number of samples ($R^2 = 1.00$), and Q in LR was stable and did not alter with n. According to the abovementioned rules, the total FPTC of SVM was corrected to $F(m^2vn^2)$ and the total FPTC of LR was corrected to $F(m^2vn)$. The updated FPTC was not affected by whether the program was run in advance.

In a future study, we plan to derive more FPTC values for algorithms. A suitable algorithm with good accuracy and low FPTC can be quickly filtered for an emergency task, which helps emergency managers make quick decisions in response to natural disasters based on the amount of remote sensing data available.

ACKNOWLEDGMENT

The authors would like to thank two reviewers for suggestions and thank Y. Zhang and J. Jiang from SIAT for valuable suggestions.

REFERENCES

[1] C. N. Koyama, H. Gokon, M. Jimbo, S. Koshimura, and M. Sato, "Disaster debris estimation using high-resolution polarimetric stereo-SAR," *ISPRS J. Photogrammetry Remote Sens.*, vol. 120, pp. 84–98, 2016.
[2] S. Liu and M. E. Hodgson, "Satellite image collection modeling for large area hazard emergency response," *ISPRS J. Photogrammetry Remote Sens.*, vol. 118, pp. 13–21, 2016.
[3] X. Zheng, F. Wang, and Z. Li, "A multi-UAV cooperative route planning methodology for 3D fine-resolution building model reconstruction," *ISPRS J. Photogrammetry Remote Sens.*, vol. 146, pp. 483–494, 2018.
[4] R. Khatami, G. Mountrakis, and S. V. Stehman, "A meta-analysis of remote sensing research on supervised pixel-based land-cover image classification processes: General guidelines for practitioners and future research," *Remote Sens. Environ.*, vol. 177, pp. 89–100, 2016.
[5] D. Marmanis, J. D. Wegner, S. Galliani, K. Schindler, M. Datcu, and U. Stilla, "Semantic segmentation of aerial images with an ensemble of CNNs," in *Proc. ISPRS Ann. Photogrammetry, Remote Sens. Spatial Inf. Sci.*, 2016, pp. 473–480.
[6] L. Yu *et al.*, "Meta-discoveries from a synthesis of satellite-based land-cover mapping research," *Int. J. Remote Sens.*, vol. 35, no. 13, pp. 4573–4588, 2014.
[7] X. Huang, J. Wang, J. Shang, C. Liao, and J. Liu, "Application of polarization signature to land cover scattering mechanism analysis and classification using multi-temporal c-band polarimetric RADARSAT-2 imagery," *Remote Sens. Environ.*, vol. 193, pp. 11–28, 2017.
[8] D. L. Donoho, Y. Tsaig, I. Drori, and J. L. Starck, "Sparse solution of underdetermined systems of linear equations by stagewise orthogonal matching pursuit," *IEEE Trans. Inf. Theory*, vol. 58, no. 2, pp. 1094–1121, Feb. 2012.
[9] T. H. Cormen, C. E. Leiserson, R. L, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009.
[10] F. A. Mianji and Y. Zhang, "Robust hyperspectral classification using relevance vector machine," *IEEE Trans. Geosci. Remote Sens.*, vol. 49, no. 6, pp. 2100–2112, Jun. 2011.
[11] A. Plaza and C.-I. Chang, "An improved N-FINDR algorithm in implementation," in *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XI*, vol. 5806, Orlando, FL, USA, SPIE, 2005.
[12] W. Yan and W. Wu, *Data structure*. Beijing, China: Tsinghua Univ. Press., (in Chinese), 1992.
[13] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, vol. 2. New Jersey, USA: Addison-Wesley, 1997.
[14] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. 13, no. 1, pp. 21–27, Jan. 1967.
[15] S. Mor-Yosef, A. Samueloff, B. Modan, D. Navot, and J. G. Schenker, "Ranking the risk factors for cesarean: Logistic regression analysis of a nationwide study," *Obstet. Gynecol.*, vol. 75, pp. 944–947, 1990.
[16] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Boca Raton, FL, USA: Chapman & Hall, 1984.
[17] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.
[18] V. N. Vapnik, *Statistical Learning Theory*. New York, NY, USA: Wiley, 1998.
[19] B. G. Batchelor, *Pattern Recognition: Ideas in Practice*. New York, NY, USA: Plenum, 1978.
[20] X. Wu *et al.*, "Top 10 algorithms in data mining," *Knowl. Inf. Syst.*, vol. 14, pp. 1–37, 2008.
[21] C. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006.
[22] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Math. Program.*, vol. 162, pp. 83–112, 2017.
[23] I. Guyon, B. Boser, and V. Vapnik, "*Automatic capacity tuning of very large VC-Dimension classifiers*," in *Proc. Adv. Neural Inf. Process. Syst.*, 1993, pp. 147–155.
[24] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.
[25] J. C. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Microsoft Research, Tech. Rep. MST-TR-98-14, 1998.
[26] R. E. Fan, P. H. Chen, and C. J. Lin, "Working set selection using second order information for training support vector machines," *J. Mach. Learn. Res.*, vol. 6, pp. 1889–1918, 2005.
[27] T. Wang, C. Z. Yan, X. Song, and J. L. Xie, "Monitoring recent trends in the area of aeolian desertified land using Landsat images in China's Xinjiang region," *ISPRS J. Photogrammetry Remote Sens.*, vol. 68, pp. 184–190, 2012.
[28] M. Drusch *et al.*, "Sentinel-2: ESA's optical high-resolution mission for GMES operational services," *Remote Sens. Environ.*, vol. 120, pp. 25–36, 2012.
[29] G. Navarro, I. Caballero, G. Silva, P. C. Parra, Á. Vázquez, and R. Caldeira, "Evaluation of forest fire on Madeira island using Sentinel-2 A MSI imagery," *Int. J. Appl. Earth Observ. Geoinformation*, vol. 58, pp. 97–106, 2017.
[30] C. Li, J. Wang, L. Wang, L. Hu, and P. Gong, "Comparison of classification algorithms and training sample sizes in urban land classification with Landsat thematic mapper imagery," *Remote Sens.*, vol. 6, pp. 964–983, 2014.
[31] C. Song, C. E. Woodcock, K. C. Seto, M. P. Lenney, and S. A. Macomber, "Classification and change detection using Landsat TM data: When and how to correct atmospheric effects?," *Remote Sens. Environ.*, vol. 75, pp. 230–244, 2001.

[32] P. Gong and P. J. Howarth, "Land-use classification of SPOT HRV data using a cover-frequency method," *Int. J. Remote Sens.*, vol. 13, pp. 1459–1471, 1992.
[33] P. Gong, D. J. Marceau, and P. J. Howarth, "A comparison of spatial feature extraction algorithms for land-use classification with SPOT HRV data," *Remote Sens. Environ.*, vol. 40, pp. 137–151, 1992.
[34] C. C. Chang and C. J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, pp. 1–27, 2011.
[35] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

Xiaorou Zheng is currently working toward the joint Ph.D. degree in computer application technology from the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China, and from the Shenzhen College of Advanced Technology, University of Chinese Academy of Sciences, Shenzhen, China.

Her current research interests include land use and land cover classification, semisupervised learning, weakly supervised learning, fully convolutional networks, and generative adversarial networks.

Jianxin Jia received the Ph.D. degree in electronic science and technology from the University of Chinese Academy of Sciences, Beijing, China, in 2018.

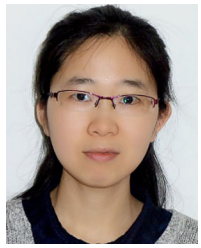
He is currently a Research Scientist with the Finnish Geospatial Research Institute. His research interests include remote sensing imaging system design, hyperspectral image processing, and application.

Shanxin Guo received the master's and Ph.D. degrees in photogrammetry and remote sensing from Wuhan University, Hubei, China, in 2011 and 2015, respectively.

He joined a visiting Ph.D. program with the Department of Geography, University of Wisconsin-Madison, Madison, WI, USA. His current research interests include digital soil mapping with remote sensing data, deep learning for land cover and land use mapping, and the Gaussian process for forest traits mapping.

Jinsong Chen received the Ph.D. degree in remote sensing and geographic information system from the Chinese Academy of Sciences, Beijing, China, in 2004.

From 2004 to 2011, he was a Postdoctoral Researcher and a Professor with the Chinese University of Hong Kong, Hong Kong. He joined the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, as the Director of the Center for Spatial Information Science and Systems. His current research interests include spatial and environmental big data processing methods and multisource spatial and environmental data assimilation and information fusion methods, and also digital soil mapping with remote sensing data, deep learning for land cover and land use mapping, and the Gaussian process for forest traits mapping.



Luyi Sun received the B.S. and M.Sc. degrees in information and communication engineering from Beijing Institute of Technology, Beijing, China, in 2009 and 2012, respectively, and the Ph.D. degree in space and climate physics from University College London, London, U.K., in 2017.

She is currently with the Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China. Her research interests include deformation monitoring of the earth surface using microwave remote sensing, as well as land cover mapping based on integration of synthetic aperture radar and multispectral imagery.

Yingfei Xiong received the B.E. degree in remote sensing science and technology from China University of Mining and Technology, Beijing, China, in 2018. He is currently working toward the master's degree in computer technology from the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China.

His research interests include image fusion and super resolution.

Wenna Xu received the master's degree in computer technology from Shenzhen Institutes of Advanced Technology, Chinese Academy of Science, Shenzhen, China, in 2020.

Her current research interests include computer vision, digital image process, and intelligent speech recognition.