Recent Advances of Large-Scale Linear Classification

This paper is a survey on development of optimization methods to construct linear classifiers suitable for large-scale applications; for some data, accuracy is close to that of nonlinear classifiers.

By Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin, *Fellow IEEE*

ABSTRACT | Linear classification is a useful tool in machine learning and data mining. For some data in a rich dimensional space, the performance (i.e., testing accuracy) of linear classifiers has shown to be close to that of nonlinear classifiers such as kernel methods, but training and testing speed is much faster. Recently, many research works have developed efficient optimization methods to construct linear classifiers and applied them to some large-scale applications. In this paper, we give a comprehensive survey on the recent development of this active research area.

KEYWORDS | Large linear classification; logistic regression; multiclass classification; support vector machines (SVMs)

I. INTRODUCTION

Linear classification is a useful tool in machine learning and data mining. In contrast to nonlinear classifiers such as kernel methods, which map data to a higher dimensional space, linear classifiers directly work on data in the original input space. While linear classifiers fail to handle some inseparable data, they may be sufficient for data in a rich dimensional space. For example, linear classifiers have shown to give competitive performances on document data with nonlinear classifiers. An important advantage of linear classification is that training and testing procedures are much more efficient. Therefore, linear classification can be very useful for some large-scale applications. Recently, the research on linear classification has been a very active topic. In this paper, we give a comprehensive survey on the recent advances.

We begin with explaining in Section II why linear classification is useful. The differences between linear and nonlinear classifiers are described. Through experiments, we demonstrate that for some data, a linear classifier achieves comparable accuracy to a nonlinear one, but both training and testing times are much shorter. Linear classifiers cover popular methods such as support vector machines (SVMs) [1], [2], logistic regression (LR),[1] and others. In Section III, we show optimization problems of these methods and discuss their differences.

An important goal of the recent research on linear classification is to develop fast optimization algorithms for training (e.g., [4]–[6]). In Section IV, we discuss issues in finding a suitable algorithm and give details of some representative algorithms. Methods such as SVM and LR were originally proposed for two-class problems. Although past works have studied their extensions to multiclass problems, the focus was on nonlinear classification. In Section V, we systematically compare methods for multiclass linear classification.

Linear classification can be further applied to many other scenarios. We investigate some examples in Section VI. In particular, we show that linear classifiers can be effectively employed to either directly or indirectly approximate nonlinear classifiers. In Section VII, we

Manuscript received June 16, 2011; revised November 24, 2011; accepted February 3, 2012. Date of publication March 30, 2012; date of current version August 16, 2012. This work was supported in part by the National Science Council of Taiwan under Grant 98-2221-E-002-136-MY3.

G.-X. Yuan was with the Department of Computer Science, National Taiwan University, Taipei 10617, Taiwan. He is currently with the University of California Davis, Davis, CA 95616 USA (e-mail: r96042@csie.ntu.edu.tw).
C.-H. Ho and **C.-J. Lin** are with the Department of Computer Science, National Taiwan University, Taipei 10617, Taiwan (e-mail: b95082@csie.ntu.edu.tw; cjlin@csie.ntu.edu.tw).

Digital Object Identifier: 10.1109/JPROC.2012.2188013

[1]It is difficult to trace the origin of logistic regression, which can be dated back to the 19th century. Interested readers may check the investigation in [3].

discuss an ongoing research topic for data larger than memory or disk capacity. Existing algorithms often fail to handle such data because of assuming that data can be stored in a single computer's memory. We present some methods which try to reduce data reading or communication time. In Section VIII, we briefly discuss related topics such as structured learning and large-scale linear regression.

Finally, Section IX concludes this survey paper.

II. WHY IS LINEAR CLASSIFICATION USEFUL?

Given training data $(y_i, \boldsymbol{x}_i) \in \{-1, +1\} \times \mathbf{R}^n$, $i = 1, \ldots, l$, where y_i is the label and $\boldsymbol{x}_i$ is the feature vector, some classification methods construct the following decision function:

$$d(\boldsymbol{x}) \equiv \boldsymbol{w}^T \phi(\boldsymbol{x}) + b \tag{1}$$

where $\boldsymbol{w}$ is the weight vector and b is an intercept, or called the bias. A *nonlinear* classifier maps each instance $\boldsymbol{x}$ to a higher dimensional vector $\phi(\boldsymbol{x})$ if data are not linearly separable. If $\phi(\boldsymbol{x}) = \boldsymbol{x}$ (i.e., data points are not mapped), we say (1) is a *linear* classifier. Because nonlinear classifiers use more features, generally they perform better than linear classifiers in terms of prediction accuracy.

For nonlinear classification, evaluating $\boldsymbol{w}^T\phi(\boldsymbol{x})$ can be expensive because $\phi(\boldsymbol{x})$ may be very high dimensional. Kernel methods (e.g., [2]) were introduced to handle such a difficulty. If $\boldsymbol{w}$ is a linear combination of training data, i.e.,

$$\boldsymbol{w} \equiv \sum_{i=1}^{l} \alpha_i \phi(\boldsymbol{x}_i), \qquad \text{for some} \quad \boldsymbol{\alpha} \in \mathbf{R}^l \tag{2}$$

and the following kernel function can be easily calculated:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$$

then the decision function can be calculated by

$$d(\boldsymbol{x}) \equiv \sum_{i=1}^{l} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b \tag{3}$$

regardless of the dimensionality of $\phi(\boldsymbol{x})$. For example

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \left(\boldsymbol{x}_i^T \boldsymbol{x}_j + 1\right)^2 \tag{4}$$

is the degree-2 polynomial kernel with

$$\phi(\boldsymbol{x}) = \Big[1, \sqrt{2}x_1, \ldots, \sqrt{2}x_n, \ldots x_1^2, \ldots x_n^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \ldots, \sqrt{2}x_{n-1}x_n\Big] \in \mathbf{R}^{(n+2)(n+1)/2}. \tag{5}$$

This kernel trick makes methods such as SVM or kernel LR practical and popular; however, for large data, the training and testing processes are still time consuming. For a kernel like (4), the cost of predicting a testing instance $\boldsymbol{x}$ via (3) can be up to $O(ln)$. In contrast, without using kernels, $\boldsymbol{w}$ is available in an explicit form, so we can predict an instance by (1). With $\phi(\boldsymbol{x}) = \boldsymbol{x}$

$$\boldsymbol{w}^T \phi(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$$

costs only $O(n)$. It is also known that training a linear classifier is more efficient. Therefore, while a linear classifier may give inferior accuracy, it often enjoys faster training and testing.

We conduct an experiment to compare linear SVM and nonlinear SVM [with the radial basis function (RBF) kernel]. Table 1 shows the accuracy and training/testing time. Generally, nonlinear SVM has better accuracy, especially for problems cod-RNA,[2] ijcnn1, covtype, webspam, and MNIST38. This result is consistent with the theoretical proof that SVM with RBF kernel and suitable parameters gives at least as good accuracy as linear kernel [10]. However, for problems with large numbers of features, i.e., real-sim, rcv1, astro-physic, yahoo-japan, and news20, the accuracy values of linear and nonlinear SVMs are similar. Regarding training and testing time, Table 1 clearly indicates that linear classifiers are at least an order of magnitude faster.

In Table 1, problems for which linear classifiers yield comparable accuracy to nonlinear classifiers are all document sets. In the area of document classification and natural language processing (NLP), a bag-of-word model is commonly used to generate feature vectors [11]. Each feature, corresponding to a word, indicates the existence

[2]In this experiment, we scaled cod-RNA feature wisely to $[-1, 1]$ interval.

Table 1 Comparison of Linear and Nonlinear Classifiers. For Linear, We Use the Software LIBLINEAR [7], While for Nonlinear We Use LIBSVM [8] (RBF Kernel). The Last Column Shows the Accuracy Difference Between Linear and Nonlinear Classifiers. Training and Testing Time Is in Seconds. The Experimental Setting Follows Exactly From [9, Sec. 4]

Data set	#instances		#features	Linear			Nonlinear (kernel)			Accuracy difference to nonlinear
				Time (s)		Testing accuracy	Time (s)		Testing accuracy	
	Training	Testing		Training	Testing		Training	Testing		
cod-RNA	59,535	271,617	8	3.1	0.05	70.71	80.2	126.02	96.67	-25.96
ijcnn1	49,990	91,701	22	1.7	0.01	92.21	26.8	20.29	98.69	-6.48
covtype	464,810	116,202	54	1.5	0.03	76.37	46,695.8	1,131.20	96.11	-19.74
webspam	280,000	70,000	254	26.8	0.04	93.35	15,681.8	853.34	99.26	-5.91
MNIST38	11,982	1,984	752	0.2	0.01	96.82	38.1	5.61	99.70	-2.88
real-sim	57,848	14,461	20,958	0.3	0.01	97.44	938.3	81.94	97.82	-0.38
rcv1	20,242	677,399	47,236	0.1	0.43	96.26	108.0	3,259.46	96.50	-0.24
astro-physic	49,896	12,473	99,757	0.3	0.01	97.09	735.7	111.59	97.31	-0.22
yahoo-japan	140,963	35,240	832,026	3.3	0.03	92.63	20,955.2	1,890.83	93.31	-0.68
news20	15,997	3,999	1,355,191	1.2	0.03	96.95	383.2	100.38	96.90	0.05

of the word in a document. Because the number of features is the same as the number of possible words, the dimensionality is huge, and the data set is often sparse. For this type of large sparse data, linear classifiers are very useful because of competitive accuracy and very fast training and testing.

III. BINARY LINEAR CLASSIFICATION METHODS

To generate a decision function (1), linear classification involves the following risk minimization problem:

$$\min_{\boldsymbol{w},b} \quad f(\boldsymbol{w},b) \equiv r(\boldsymbol{w}) + C\sum_{i=1}^{l} \xi(\boldsymbol{w},b;\boldsymbol{x}_i,y_i) \tag{6}$$

where $r(\boldsymbol{w})$ is the regularization term and $\xi(\boldsymbol{w},b;\boldsymbol{x},y)$ is the loss function associated with the observation $(y,\boldsymbol{x})$. Parameter $C > 0$ is user specified for balancing $r(\boldsymbol{w})$ and the sum of losses.

Following the discussion in Section II, linear classification is often applied to data with many features, so the bias term b may not be needed in practice. Experiments in [12] and [13] on document data sets showed similar performances with/without the bias term. In the rest of this paper, we omit the bias term b, so (6) is simplified to

$$\min_{\boldsymbol{w}} \quad f(\boldsymbol{w}) \equiv r(\boldsymbol{w}) + C\sum_{i=1}^{l} \xi(\boldsymbol{w};\boldsymbol{x}_i,y_i) \tag{7}$$

and the decision function becomes $d(\boldsymbol{x}) \equiv \boldsymbol{w}^T\boldsymbol{x}$.

A. Support Vector Machines and Logistic Regression

In (7), the loss function is used to penalize a wrongly classified observation $(\boldsymbol{x},y)$. There are three common loss functions considered in the literature of linear classification

$$\xi_{\mathrm{L1}}(\boldsymbol{w};\boldsymbol{x},y) \equiv \max(0, 1-y\boldsymbol{w}^T\boldsymbol{x}) \tag{8}$$

$$\xi_{\mathrm{L2}}(\boldsymbol{w};\boldsymbol{x},y) \equiv \max(0, 1-y\boldsymbol{w}^T\boldsymbol{x})^2 \tag{9}$$

$$\xi_{\mathrm{LR}}(\boldsymbol{w};\boldsymbol{x},y) \equiv \log\left(1+e^{-y\boldsymbol{w}^T\boldsymbol{x}}\right). \tag{10}$$

Equations (8) and (9) are referred to as L1 and L2 losses, respectively. Problem (7) using (8) and (9) as the loss function is often called L1-loss and L2-loss SVM, while problem (7) using (10) is referred to as logistic regression (LR). Both SVM and LR are popular classification methods. The three loss functions in (8)–(10) are all convex and nonnegative. L1 loss is not differentiable at the point $y\boldsymbol{w}^T\boldsymbol{x} = 1$, while L2 loss is differentiable, but not twice differentiable [14]. For logistic loss, it is twice differentiable. Fig. 1 shows that these three losses are increasing functions of $-y\boldsymbol{w}^T\boldsymbol{x}$. They slightly differ in the amount of penalty imposed.

B. L1 and L2 Regularization

A classifier is used to predict the label y for a hidden (testing) instance $\boldsymbol{x}$. Overfitting training data to minimize

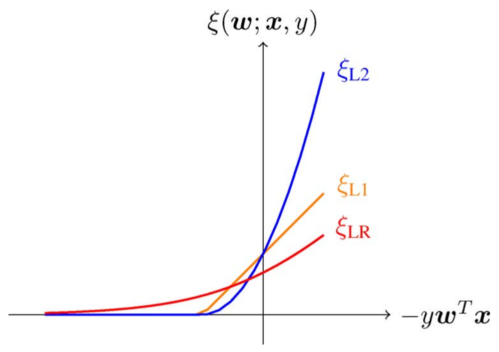


Fig. 1. ***Three loss functions:*** ξ_{L1}, ξ_{L2}, ***and*** ξ_{LR}. ***The*** x***-axis is*** $-yw^Tx$.

the training loss may not imply that the classifier gives the best testing accuracy. The concept of regularization is introduced to prevent from overfitting observations. The following L2 and L1 regularization terms are commonly used:

$$r_{\mathrm{L2}}(\boldsymbol{w}) \equiv \frac{1}{2}\|\boldsymbol{w}\|_2^2 = \frac{1}{2}\sum_{j=1}^{n} w_j^2 \tag{11}$$

and

$$r_{\mathrm{L1}}(\boldsymbol{w}) \equiv \|\boldsymbol{w}\|_1 = \sum_{j=1}^{n} |w_j|. \tag{12}$$

Problem (7) with L2 regularization and L1 loss is the standard SVM proposed in [1]. Both (11) and (12) are convex and separable functions. The effect of regularization on a variable is to push it toward zero. Then, the search space of $\boldsymbol{w}$ is more confined and overfitting may be avoided. It is known that an L1-regularized problem can generate a sparse model with few nonzero elements in $\boldsymbol{w}$. Note that $w^2/2$ becomes more and more flat toward zero, but $|w|$ is uniformly steep. Therefore, an L1-regularized variable is easier to be pushed to zero, but a caveat is that (12) is not differentiable. Because nonzero elements in $\boldsymbol{w}$ may correspond to useful features [15], L1 regularization can be applied for feature selection. In addition, less memory is needed to store $\boldsymbol{w}$ obtained by L1 regularization. Regarding testing accuracy, comparisons such as [13, Suppl. Mater. Sec. D] show that L1 and L2 regularizations generally give comparable performance.

In statistics literature, a model related to L1 regularization is LASSO [16]

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \sum_{i=1}^{l} \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i) \\ \text{subject to} \quad & \|\boldsymbol{w}\|_1 \le K \end{aligned} \tag{13}$$

where $K > 0$ is a parameter. This optimization problem is equivalent to (7) with L1 regularization. That is, for a given C in (7), there exists K such that (13) gives the same solution as (7). The explanation for this relationship can be found in, for example, [17].

Any combination of the above-mentioned two regularizations and three loss functions has been well studied in linear classification. Of them, L2-regularized L1/L2-loss SVM can be geometrically interpreted as maximum margin classifiers. L1/L2-regularized LR can be interpreted in a Bayesian view by maximizing the posterior probability with Laplacian/Gaussian prior of $\boldsymbol{w}$.

A convex combination of L1 and L2 regularizations forms the elastic net [18]

$$r_e(\boldsymbol{w}) \equiv \lambda\|\boldsymbol{w}\|_2^2 + (1-\lambda)\|\boldsymbol{w}\|_1 \tag{14}$$

where $\lambda \in [0, 1)$. The elastic net is used to break the following limitations of L1 regularization. First, L1 regularization term is not strictly convex, so the solution may not be unique. Second, for two highly correlated features, the solution obtained by L1 regularization may select only one of these features. Consequently, L1 regularization may discard the group effect of variables with high correlation [18].

IV. TRAINING TECHNIQUES

To obtain the model $\boldsymbol{w}$, in the training phase we need to solve the convex optimization problem (7). Although many convex optimization methods are available, for large linear classification, we must carefully consider some factors in designing a suitable algorithm. In this section, we first discuss these design issues and follow by showing details of some representative algorithms.

A. Issues in Finding Suitable Algorithms

- *Data property.* Algorithms that are efficient for some data sets may be slow for others. We must take data properties into account in selecting algorithms. For example, we can check if the number of instances is much larger than features, or *vice versa*. Other useful properties include the number of nonzero feature values, feature distribution, feature correlation, etc.
- *Optimization formulation.* Algorithm design is strongly related to the problem formulation. For example, most unconstrained optimization techniques can be applied to L2-regularized logistic regression, while specialized algorithms may be needed for the nondifferentiable L1-regularized problems.

  In some situations, by reformulation, we are able to transform a nondifferentiable problem to be differentiable. For example, by letting $\boldsymbol{w} = \boldsymbol{w}^+ - \boldsymbol{w}^-$ $(\boldsymbol{w}^+, \boldsymbol{w}^- \ge \mathbf{0})$, L1-regularized classifiers can be written as

$$\begin{aligned} \min_{\boldsymbol{w}^+, \boldsymbol{w}^-} \quad & \sum_{j=1}^{n} w_j^+ + \sum_{j=1}^{n} w_j^- + \sum_{i=1}^{l} \xi(\boldsymbol{w}^+ - \boldsymbol{w}^-; \boldsymbol{x}_i, y_i) \\ \text{subject to} \quad & w_j^+, w_j^- \ge 0, \qquad j = 1, \ldots, n. \end{aligned} \tag{15}$$

However, there is no guarantee that solving a differentiable form is faster. Recent comparisons [13] show that for L1-regularized classifiers, methods directly minimizing the nondifferentiable form are often more efficient than those solving (15).

- *Solving primal or dual problems.* Problem (7) has n variables. In some applications, the number of instances l is much smaller than the number of features n. By Lagrangian duality, a dual problem of (7) has l variables. If $l \ll n$, solving the dual form may be easier due to the smaller number of variables. Further, in some situations, the dual problem possesses nice properties not in the primal form. For example, the dual problem of the standard SVM (L2-regularized L1-loss SVM) is the following quadratic program[3]:

$$\min_{\boldsymbol{\alpha}} \quad f^D(\boldsymbol{\alpha}) \equiv \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha}$$
$$\text{subject to} \quad 0 \le \alpha_i \le C \qquad \forall i = 1, \dots, l \quad (16)$$

where $Q_{ij} \equiv y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$. Although the primal objective function is nondifferentiable because of the L1 loss, in (16), the dual objective function is smooth (i.e., derivatives of all orders are available). Hence, solving the dual problem may be easier than primal because we can apply differentiable optimization techniques. Note that the primal optimal $\boldsymbol{w}$ and the dual optimal $\boldsymbol{\alpha}$ satisfy the relationship (2),[4] so solving primal and dual problems leads to the same decision function.

Dual problems come with another nice property that each variable α_i corresponds to a training instance $(y_i, \boldsymbol{x}_i)$. In contrast, for primal problems, each variable w_i corresponds to a feature. Optimization methods which update some variables at a time often need to access the corresponding instances (if solving dual) or the corresponding features (if solving primal). In practical applications, instance-wise data storage is more common than feature-wise storage. Therefore, a dual-based algorithm can directly work on the input data without any transformation.

Unfortunately, the dual form may not be always easier to solve. For example, the dual form of L1-regularized problems involves general linear constraints rather than bound constraints in (16), so solving primal may be easier.

- *Using low-order or high-order information.* Low-order methods, such as gradient or subgradient methods, have been widely considered in large-scale training. They characterize low-cost update, low-memory requirement, and slow convergence. In classification tasks, slow convergence may not be a serious concern because a loose solution of (7) may already give similar testing performances to that by an accurate solution.

High-order methods such as Newton methods often require the smoothness of the optimization problems. Further, the cost per step is more expensive; sometimes a linear system must be solved. However, their convergence rate is superior. These high-order methods are useful for applications needing an accurate solution of problem (7). Some (e.g., [20]) have tried a hybrid setting by using low-order methods in the beginning and switching to higher order methods in the end.
- *Cost of different types of operations.* In a real-world computer, not all types of operations cost equally. For example, exponential and logarithmic operations are much more expensive than multiplication and division. For training large-scale LR, because exp / log operations are required, the cost of this type of operations may accumulate faster than that of other types. An optimization method which can avoid intensive exp / log evaluations is potentially efficient; see more discussion in, for example, [12], [21], and [22].
- *Parallelization.* Most existing training algorithms are inherently sequential, but a parallel algorithm can make good use of the computational power in a multicore machine or a distributed system. However, the communication cost between different cores or nodes may become a new bottleneck. See more discussion in Section VII.

Earlier developments of optimization methods for linear classification tend to focus on data with few features. By taking this property, they are able to easily train millions of instances [23]. However, these algorithms may not be suitable for sparse data with both large numbers of instances and features, for which we show in Section II that linear classifiers often give competitive accuracy with nonlinear classifiers. Many recent studies have proposed algorithms for such data. We list some of them (and their software name if any) according to regularization and loss functions used.

- *L2-regularized L1-loss SVM*: Available approaches include, for example, cutting plane methods for the primal form (SVM$^{\text{perf}}$ [4], OCAS [24], and BMRM [25]), a stochastic (sub)gradient descent method for the primal form (Pegasos [5], and SGD [26]), and a coordinate descent method for the dual form (LIBLINEAR [6]).
- *L2-regularized L2-loss SVM*: Existing methods for the primal form include a coordinate descent method [21], a Newton method [27], and a trust

[3]Because the bias term b is not considered, therefore, different from the dual problem considered in SVM literature, an inequality constraint $\sum y_i\alpha_i = 0$ is absent from (16).

[4]However, we do not necessarily need the dual problem to get (2). For example, the reduced SVM [19] directly assumes that $\boldsymbol{w}$ is the linear combination of a subset of data.

region Newton method (LIBLINEAR [28]). For the dual problem, a coordinate descent method is in the software LIBLINEAR [6].

- *L2-regularized LR*: Most unconstrained optimization methods can be applied to solve the primal problem. An early comparison on small-scale data is [29]. Existing studies for large sparse data include iterative scaling methods [12], [30], [31], a truncated Newton method [32], and a trust region Newton method (LIBLINEAR [28]). Few works solve the dual problem. One example is a coordinate descent method (LIBLINEAR [33]).
- *L1-regularized L1-loss SVM*: It seems no studies have applied L1-regularized L1-loss SVM on large sparse data although some early works for data with either few features or few instances are available [34]–[36].
- *L1-regularized L2-loss SVM*: Some proposed methods include a coordinate descent method (LIBLINEAR [13]) and a Newton-type method [22].
- *L1-regularized LR*: Most methods solve the primal form, for example, an interior-point method (l1_logreg [37]), (block) coordinate descent methods (BBR [38] and CGD [39]), a quasi-Newton method (OWL-QN [40]), Newton-type methods (GLMNET [41] and LIBLINEAR [22]), and a Nesterov's method (SLEP [42]). Recently, an augmented Lagrangian method (DAL [43]) was proposed for solving the dual problem. Comparisons of methods for L1-regularized LR include [13] and [44].

In the rest of this section, we show details of some optimization algorithms. We select them not only because they are popular but also because many design issues discussed earlier can be covered.

B. Example: A Subgradient Method (Pegasos With Deterministic Settings)

Shalev-Shwartz *et al.* [5] proposed a method Pegasos for solving the primal form of L2-regularized L1-loss SVM. It can be used for batch and online learning. Here we discuss only the deterministic setting and leave the stochastic setting in Section VII-A.

Given a training subset B, at each iteration, Pegasos approximately solves the following problem:

$$\min_{\boldsymbol{w}} \quad f(\boldsymbol{w};B) \equiv \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i\in B}\max(0, 1-y_i\boldsymbol{w}^T\boldsymbol{x}).$$

Here, for the deterministic setting, B is the whole training set. Because L1 loss is not differentiable, Pegasos takes the following subgradient direction of $f(\boldsymbol{w};B)$:

$$\nabla^S f(\boldsymbol{w};B) \equiv \boldsymbol{w} - C\sum_{i\in B^+} y_i\boldsymbol{x}_i \tag{17}$$

where $B^+ \equiv \{i | i\in B, 1-y_i\boldsymbol{w}^T\boldsymbol{x}_i > 0\}$, and updates $\boldsymbol{w}$ by

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\nabla^S f(\boldsymbol{w};B) \tag{18}$$

where $\eta = (Cl)/k$ is the learning rate and k is the iteration index. Different from earlier subgradient descent methods, after the update by (18), Pegasos further projects $\boldsymbol{w}$ onto the ball set $\{\boldsymbol{w} | \|\boldsymbol{w}\|_2 \le \sqrt{Cl}\}$.[5] That is

$$\boldsymbol{w} \leftarrow \min\left(1, \frac{\sqrt{Cl}}{\|\boldsymbol{w}\|_2}\right)\boldsymbol{w}. \tag{19}$$

We show the overall procedure of Pegasos in Algorithm 1.

Algorithm 1: Pegasos for L2-regularized L1-loss SVM (deterministic setting for batch learning) [5]

```
1) Given $\boldsymbol{w}$ such that $\|\boldsymbol{w}\|_2 \le \sqrt{Cl}$.
2) For $k = 1, 2, 3, \ldots$
   a) Let $B = \{(y_i, \boldsymbol{x}_i)\}_{i=1}^{l}$.
   b) Compute the learning rate $\eta = (Cl)/k$.
   c) Compute $\nabla^S f(\boldsymbol{w};B)$ by (17).
   d) $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\nabla^S f(\boldsymbol{w};B)$.
   e) Project $\boldsymbol{w}$ by (19) to ensure $\|\boldsymbol{w}\|_2 \le \sqrt{Cl}$.
```

For convergence, it is proved that in $O(1/\epsilon)$ iterations, Pegasos achieves an average ϵ-accurate solution. That is

$$f\left(\left(\sum_{k=1}^{T}\boldsymbol{w}^k\right)\Big/T\right) - f(\boldsymbol{w}^*) \le \epsilon$$

where $\boldsymbol{w}^k$ is the kth iterate and $\boldsymbol{w}^*$ is the optimal solution.

Pegasos has been applied in many studies. One implementation issue is that information obtained in the algorithm cannot be directly used for designing a suitable stopping condition.

C. Example: Trust Region Newton Method (TRON)

Trust region Newton method (TRON) is an effective approach for unconstrained and bound-constrained optimization. In [28], it applies the setting in [45] to solve (7) with L2 regularization and differentiable losses.

[5]The optimal solution of $f(\boldsymbol{w})$ is proven to be in the ball set $\{\boldsymbol{w} | \|\boldsymbol{w}\|_2 \le \sqrt{Cl}\}$; see [5, Th. 1].

At each iteration, given an iterate $\boldsymbol{w}$, a trust region interval Δ, and a quadratic model

$$q(\boldsymbol{d}) \equiv \nabla f(\boldsymbol{w})^T \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T \nabla^2 f(\boldsymbol{w}) \boldsymbol{d} \tag{20}$$

as an approximation of $f(\boldsymbol{w}+\boldsymbol{d}) - f(\boldsymbol{w})$, TRON finds a truncated Newton step confined in the trust region by approximately solving the following subproblem:

$$\min_{\boldsymbol{d}} \; q(\boldsymbol{d}) \quad \text{subject to} \quad \|\boldsymbol{d}\|_2 \leq \Delta. \tag{21}$$

Then, by checking the ratio

$$\sigma \equiv \frac{f(\boldsymbol{w}+\boldsymbol{d}) - f(\boldsymbol{w})}{q(\boldsymbol{d})} \tag{22}$$

of actual function reduction to estimated function reduction, TRON decides if $\boldsymbol{w}$ should be updated and then adjusts Δ. A large enough σ indicates that the quadratic model $q(\boldsymbol{d})$ is close to $f(\boldsymbol{w}+\boldsymbol{d}) - f(\boldsymbol{w})$, so TRON updates $\boldsymbol{w}$ to be $\boldsymbol{w}+\boldsymbol{d}$ and slightly enlarges the trust region interval Δ for the next iteration. Otherwise, the current iterate $\boldsymbol{w}$ is unchanged and the trust region interval Δ shrinks by multiplying a factor less than one. The overall procedure of TRON is presented in Algorithm 2.

Algorithm 2: TRON for L2-regularized LR and L2-loss SVM [28]

```
1) Given $\boldsymbol{w}$, $\Delta$, and $\sigma_0$.
2) For $k = 1, 2, 3, \ldots$
    a) Find an approximate solution $\boldsymbol{d}$ of (21) by the
       conjugate gradient method.
    b) Check the ratio $\sigma$ in (22).
    c) If $\sigma > \sigma_0$
           $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{d}$.
    d) Adjust $\Delta$ according to $\sigma$.
```

If the loss function is not twice differentiable (e.g., L2 loss), we can use generalized Hessian [14] as $\nabla^2 f(\boldsymbol{w})$ in (20).

Some difficulties of applying Newton methods to linear classification include that $\nabla^2 f(\boldsymbol{w})$ may be a huge n by n matrix and solving (21) is expensive. Fortunately, $\nabla^2 f(\boldsymbol{w})$ of linear classification problems takes the following special form:

$$\nabla^2 f(\boldsymbol{w}) = \mathcal{I} + CX^T D_{\boldsymbol{w}} X$$

where $\mathcal{I}$ is an identity matrix, $X \equiv [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l]^T$, and $D_{\boldsymbol{w}}$ is a diagonal matrix. In [28], a conjugate gradient method is applied to solve (21), where the main operation is the product between $\nabla^2 f(\boldsymbol{w})$ and a vector $\boldsymbol{v}$. By

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{v} = \boldsymbol{v} + C\left(X^T(D_{\boldsymbol{w}}(X\boldsymbol{v}))\right) \tag{23}$$

the Hessian matrix $\nabla^2 f(\boldsymbol{w})$ need not be stored.

Because of using high-order information (Newton directions), TRON gives fast quadratic local convergence. It has been extended to solve L1-regularized LR and L2-loss SVM in [13] by reformulating (7) to a bound-constrained optimization problem in (15).

D. Example: Solving Dual SVM by Coordinate Descent Methods (Dual-CD)

Hsieh *et al.* [6] proposed a coordinate descent method for the dual L2-regularized linear SVM in (16). We call this algorithm Dual-CD. Here, we focus on L1-loss SVM, although the same method has been applied to L2-loss SVM in [6].

A coordinate descent method sequentially selects one variable for update and fixes others. To update the ith variable, the following one-variable problem is solved:

$$\begin{aligned} \min_{d} \quad & f^D(\boldsymbol{\alpha} + d\boldsymbol{e}_i) - f^D(\boldsymbol{\alpha}) \\ \text{subject to} \quad & 0 \leq \alpha_i + d \leq C \end{aligned}$$

where $f(\boldsymbol{\alpha})$ is defined in (16), $\boldsymbol{e}_i = [\underbrace{0, \ldots, 0}_{i-1}, 1, 0, \ldots, 0]^T$, and

$$f^D(\boldsymbol{\alpha} + d\boldsymbol{e}_i) - f^D(\boldsymbol{\alpha}) = \frac{1}{2}Q_{ii}d^2 + \nabla_i f^D(\boldsymbol{\alpha})d.$$

This simple quadratic function can be easily minimized. After considering the constraint, a simple update rule for α_i is

$$\alpha_i \leftarrow \min\left(\max\left(\alpha_i - \frac{\nabla_i f^D(\boldsymbol{\alpha})}{Q_{ii}}, 0\right), C\right). \tag{24}$$

From (24), Q_{ii} and $\nabla_i f^D(\boldsymbol{\alpha})$ are our needs. The diagonal entries of Q, $Q_{ii}, \forall i$, are computed only once initially, but

$$\nabla_i f^D(\boldsymbol{\alpha}) = (Q\boldsymbol{\alpha})_i - 1 = \sum_{t=1}^{l} \left(y_i y_t \boldsymbol{x}_i^T \boldsymbol{x}_t\right)\alpha_t - 1 \tag{25}$$

requires $O(nl)$ cost for l inner products $\boldsymbol{x}_i^T\boldsymbol{x}_t, \forall t = 1, \ldots, l$. To make coordinate descent methods viable for large linear classification, a crucial step is to maintain

$$\boldsymbol{u} \equiv \sum_{t=1}^{l} y_t \alpha_t \boldsymbol{x}_t \tag{26}$$

so that (25) becomes

$$\nabla_i f^D(\boldsymbol{\alpha}) = (Q\boldsymbol{\alpha})_i - 1 = y_i \boldsymbol{u}^T \boldsymbol{x}_i - 1. \tag{27}$$

If $\boldsymbol{u}$ is available through the training process, then the cost $O(nl)$ in (25) is significantly reduced to $O(n)$. The remaining task is to maintain $\boldsymbol{u}$. Following (26), if $\bar{\alpha}_i$ and α_i are values before and after the update (24), respectively, then we can easily maintain $\boldsymbol{u}$ by the following $O(n)$ operation:

$$\boldsymbol{u} \leftarrow \boldsymbol{u} + y_i(\alpha_i - \bar{\alpha}_i)\boldsymbol{x}_i. \tag{28}$$

Therefore, the total cost for updating an α_i is $O(n)$. The overall procedure of the coordinate descent method is in Algorithm 3.

Algorithm 3: A coordinate descent method for L2-regularized L1-loss SVM [6]

```
1) Given $\boldsymbol{\alpha}$ and the corresponding $\boldsymbol{u} = \sum_{i=1}^{l} y_i \alpha_i \boldsymbol{x}_i$.
2) Compute $Q_{ii}, \forall i = 1, \ldots, l$.
3) For $k = 1, 2, 3, \ldots$
   • For $i = 1, \ldots, l$
       a) Compute $G = y_i \boldsymbol{u}^T \boldsymbol{x}_i - 1$ in (27).
       b) $\bar{\alpha}_i \leftarrow \alpha_i$.
       c) $\alpha_i \leftarrow \min(\max(\alpha_i - G/Q_{ii}, 0), C)$.
       d) $\boldsymbol{u} \leftarrow \boldsymbol{u} + y_i(\alpha_i - \bar{\alpha}_i)\boldsymbol{x}_i$.
```

The vector $\boldsymbol{u}$ defined in (26) is in the same form as $\boldsymbol{w}$ in (2). In fact, as $\boldsymbol{\alpha}$ approaches a dual optimal solution, $\boldsymbol{u}$ will converge to the primal optimal $\boldsymbol{w}$ following the primal–dual relationship.

The linear convergence of Algorithm 3 is established in [6] using techniques in [46]. The authors propose two implementation tricks to speed up the convergence. First, instead of a sequential update, they repeatedly permute $\{1, \ldots, l\}$ to decide the order. Second, similar to the shrinking technique used in training nonlinear SVM [47], they identify some bounded variables which may already be optimal and remove them during the optimization procedure. Experiments in [6] show that for large sparse data, Algorithm 3 is much faster than TRON in the early stage. However, it is less competitive if the parameter C is large.

Algorithm 3 is very related to popular decomposition methods used in training nonlinear SVM (e.g., [8] and [47]). These decomposition methods also update very few variables at each step, but use more sophisticated schemes for selecting variables. The main difference is that for linear SVM, we can define $\boldsymbol{u}$ in (26) because $\boldsymbol{x}_i, \forall i$ are available. For nonlinear SVM, $\nabla_i f^D(\boldsymbol{w})$ in (25) needs $O(nl)$ cost for calculating l kernel elements. This difference between $O(n)$ and $O(nl)$ is similar to that in the testing phase discussed in Section II.

E. Example: Solving L1-Regularized Problems by Combining Newton and Coordinate Descent Methods (newGLMNET)

GLMNET proposed by Friedman *et al.* [41] is a Newton method for L1-regularized minimization. An improved version newGLMNET [22] is proposed for large-scale training.

Because the 1-norm term is not differentiable, we represents $f(\boldsymbol{w})$ as the sum of two terms $\|\boldsymbol{w}\|_1 + L(\boldsymbol{w})$, where

$$L(\boldsymbol{w}) \equiv C \sum_{i=1}^{l} \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i).$$

At each iteration, newGLMNET considers the second-order approximation of $L(\boldsymbol{w})$ and solves the following problem:

$$\min_{\boldsymbol{d}} \quad q(\boldsymbol{d}) \equiv \|\boldsymbol{w}+\boldsymbol{d}\|_1 - \|\boldsymbol{w}\|_1 + \nabla L(\boldsymbol{w})^T\boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^T H \boldsymbol{d} \tag{29}$$

where $H \equiv \nabla^2 L(\boldsymbol{w}) + \nu\mathcal{I}$ and ν is a small number to ensure H to be positive definite.

Although (29) is similar to (21), its optimization is more difficult because of the 1-norm term. Thus, newGLMNET further breaks (29) to subproblems by a coordinate descent procedure. In a setting similar to the

Table 2 A Comparison of the Four Methods in Section IV-B–E

	Pegasos	TRON	Dual-CD	newGLMNET
primal-/dual-based	primal	primal	dual	primal
low/high order	low	high	low	high
data access	instance-wisely	both	instance-wisely	feature-wisely

method in Section IV-D, each time a one-variable function is minimized

$$q(\boldsymbol{d}+z\boldsymbol{e}_j)-q(\boldsymbol{d})=|w_j+d_j+z|-|w_j+d_j|+G_jz+\frac{1}{2}H_{jj}z^2 \tag{30}$$

where $G\equiv\nabla L(\boldsymbol{w})+H\boldsymbol{d}$. This one-variable function (30) has a simple closed-form minimizer (see [48], [49], and [13, App. B])

$$z=\begin{cases}-\frac{G_j+1}{H_{jj}}, & \text{if } G_j+1\le H_{jj}(w_j+d_j)\\ -\frac{G_j-1}{H_{jj}}, & \text{if } G_j-1\ge H_{jj}(w_j+d_j)\\ -(w_j+d_j), & \text{otherwise.}\end{cases}$$

At each iteration of newGLMNET, the coordinate descent method does not solve problem (29) exactly. Instead, newGLMNET designs an adaptive stopping condition so that initially problem (29) is solved loosely and in the final iterations, (29) is more accurately solved.

After an approximate solution $\boldsymbol{d}$ of (29) is obtained, we need a line search procedure to ensure the sufficient function decrease. It finds $\lambda\in(0,1]$ such that

$$f(\boldsymbol{w}+\lambda\boldsymbol{d})-f(\boldsymbol{w})\le\sigma\lambda\left(\|\boldsymbol{w}+\boldsymbol{d}\|_1-\|\boldsymbol{w}\|_1+\nabla L(\boldsymbol{w})^T\boldsymbol{d}\right) \tag{31}$$

where $\sigma\in(0,1)$. The overall procedure of newGLMNET is in Algorithm 4.

Algorithm 4: newGLMNET for L1-regularized minimization [22]

```
1) Given $\boldsymbol{w}$. Given $0<\beta,\sigma<1$.
2) For $k=1,2,3,\ldots$
    a) Find an approximate solution $\boldsymbol{d}$ of (29) by a coordinate descent method.
    b) Find $\lambda=\max\{1,\beta,\beta^2,\ldots\}$ such that (31) holds.
    c) $\boldsymbol{w}\leftarrow\boldsymbol{w}+\lambda\boldsymbol{d}$.
```

Due to the adaptive setting, in the beginning newGLMNET behaves like a coordinate descent method, which is able to quickly obtain an approximate $\boldsymbol{w}$; however, in the final stage, the iterate $\boldsymbol{w}$ converges quickly because a Newton step is taken. Recall in Section IV-A, we mentioned that exp / log operations are more expensive than basic operations such as multiplication/division. Because (30) does not involve any exp / log operation, we successfully achieve that time spent on exp / log operations is only a small portion of the whole procedure. In addition, newGLMNET is an example of accessing data feature wisely; see details in [22] about how G_j in (30) is updated.

F. A Comparison of the Four Examples

The four methods discussed in Sections IV-B–E differ in various aspects. By considering design issues mentioned in Section IV-A, we compare these methods in Table 2. We point out that three methods are primal based, but one is dual based. Next, both Pegasos and Dual-CD use only low-order information (subgradient and gradient), but TRON and newGLMNET employ high-order information by Newton directions. Also, we check how data instances are accessed. Clearly, Pegasos and Dual-CD instance wisely access data, but we have mentioned in Section IV-E that newGLMNET must employ a feature wisely setting. Interestingly, TRON can use both because in (23), matrix–vector products can be conducted by accessing data instance wisely or feature wisely.

We analyze the complexity of the four methods by showing the cost at the kth iteration:

- Pegasos: $O(|B^+|n)$;
- TRON: $\#\text{CG\_iter}\times O(ln)$;
- Dual-CD: $O(ln)$;
- newGLMNET: $\#\text{CD\_iter}\times O(ln)$.

The cost of Pegasos and TRON easily follows from (17) and (23), respectively. For Dual-CD, both (27) and (28) cost $O(n)$, so one iteration of going through all variables is $O(nl)$. For newGLMNET, see details in [22]. We can clearly see that each iteration of Pegasos and Dual-CD is cheaper because of using low-order information. However, they need more iterations than high-order methods in order to accurately solve the optimization problem.

V. MULTICLASS LINEAR CLASSIFICATION

Most classification methods are originally proposed to solve a two-class problem; however, extensions of these methods to multiclass classification have been studied. For nonlinear SVM, some works (e.g., [50] and [51]) have comprehensively compared different multiclass solutions. In contrast, few studies have focused on multiclass linear classification. This section introduces and compares some commonly used methods.

A. Solving Several Binary Problems

Multiclass classification can be decomposed to several binary classification problems. One-against-rest and one-against-one methods are two of the most common decomposition approaches. Studies that broadly discussed various approaches of decomposition include, for example, [52] and [53].

- *One-against-rest method.* If there are k classes in the training data, the one-against-rest method [54] constructs k binary classification models. To obtain the mth model, instances from the mth class of the training set are treated as positive, and all other instances are negative. Then, the weight vector $\boldsymbol{w}_m$ for the mth model can be generated by any linear classifier.

  After obtaining all k models, we say an instance $\boldsymbol{x}$ is in the mth class if the decision value (1) of the mth model is the largest, i.e.,

$$\text{class of } \boldsymbol{x} \equiv \arg \max_{m=1,\dots,k} \boldsymbol{w}_m^T \boldsymbol{x}. \tag{32}$$

  The cost for testing an instance is $O(nk)$.
- *One-against-one method.* One-against-one method [55] solves $k(k-1)/2$ binary problems. Each binary classifier constructs a model with data from one class as positive and another class as negative. Since there is $k(k-1)/2$ combination of two classes, $k(k-1)/2$ weight vectors are constructed: $\boldsymbol{w}_{1,2}, \boldsymbol{w}_{1,3}, \dots, \boldsymbol{w}_{1,k}, \boldsymbol{w}_{2,3}, \dots, \boldsymbol{w}_{(k-1),k}$.

  There are different methods for testing. One approach is by voting [56]. For a testing instance $\boldsymbol{x}$, if model (i,j) predicts $\boldsymbol{x}$ as in the ith class, then a counter for the ith class is added by one; otherwise, the counter for the jth class is added. Then, we say $\boldsymbol{x}$ is in the ith class if the ith counter has the largest value. Other prediction methods are similar though they differ in how to use the $k(k-1)/2$ decision values; see some examples in [52] and [53].

  For linear classifiers, one-against-one method is shown to give better testing accuracy than one-against-rest method [57]. However, it requires $O(k^2n)$ spaces for storing models and $O(k^2n)$ cost for testing an instance; both are more expensive than the one-against-rest method. Interestingly, for nonlinear classifiers via kernels, one-against-one method does not have such disadvantages [50].

  DAGSVM [58] is the same as one-against-one method but it attempts to reduce the testing cost. Starting with a candidate set of all classes, this method sequentially selects a pair of classes for prediction and removes one of the two. That is, if a binary classifier of class i and j predicts i, then j is removed from the candidate set. Alternatively, a prediction of class j will cause i to be removed. Finally, the only remained class is the predicted result. For any pair (i,j) considered, the true class may be neither i nor j. However, it does not matter which one is removed because all we need is that if the true class is involved in a binary prediction, it is the winner. Because classes are sequentially removed, only $k-1$ models are used. The testing time complexity of DAGSVM is thus $O(nk)$.

B. Considering All Data at Once

In contrast to using many binary models, some have proposed solving a single optimization problem for multiclass classification [59]–[61]. Here we discuss details of Crammer and Singer's approach [60]. Assume class labels are $1,\dots,k$. They consider the following optimization problem:

$$\min_{\boldsymbol{w}_1,\dots,\boldsymbol{w}_k} \quad \frac{1}{2}\sum_{m=1}^{k} \|\boldsymbol{w}_m\|_2^2 + C\sum_{i=1}^{l} \xi_{\text{CS}}\left(\{\boldsymbol{w}_m\}_{m=1}^{k}; \boldsymbol{x}_i, y_i\right) \tag{33}$$

where

$$\xi_{\text{CS}}\left(\{\boldsymbol{w}_m\}_{m=1}^{k}; \boldsymbol{x}, y\right) \equiv \max_{m \neq y} \max\left(0, 1 - (\boldsymbol{w}_y - \boldsymbol{w}_m)^T \boldsymbol{x}\right). \tag{34}$$

The setting is like combining all binary models of the one-against-rest method. There are k weight vectors $\boldsymbol{w}_1,\dots,\boldsymbol{w}_k$ for k classes. In the loss function (34), for each m, $\max(0, 1-(\boldsymbol{w}_{y_i} - \boldsymbol{w}_m)^T \boldsymbol{x}_i)$ is similar to the L1 loss in (8) for binary classification. Overall, we hope that the decision value of $\boldsymbol{x}_i$ by the model $\boldsymbol{w}_{y_i}$ is at least one larger than the values by other models. For testing, the decision function is also (32).

Early works of this method focus on the nonlinear (i.e., kernel) case [50], [60], [62]. A study for linear classification is in [63], which applies a coordinate descent method to solve the dual problem of (33). The idea is similar to the method in Section IV-D; however, at each step, a larger subproblem of k variables is solved. A nice property of this k-variable subproblem is that it has a closed-form solution. Experiments in [63] show that solving (33) gives slightly better accuracy than one-against-rest method, but the training time is competitive. This result is different from the nonlinear case, where the longer training time than one-against-rest and one-against-one methods has made the approach of solving one single optimization problem less practical [50]. A careful implementation of the approach in [63] is given in [7, App. E].

C. Maximum Entropy

Maximum entropy (ME) [64] is a generalization of logistic regression for multiclass problems[6] and a special case of conditional random fields [65] (see Section VIII-A). It is widely applied by NLP applications. We still assume class labels $1, \ldots, k$ for an easy comparison to (33) in our subsequent discussion. ME models the following conditional probability function of label y given data $\boldsymbol{x}$:

$$P(y|\boldsymbol{x}) \equiv \frac{\exp\left(\boldsymbol{w}_y^T \boldsymbol{x}\right)}{\sum_{m=1}^{k} \exp\left(\boldsymbol{w}_m^T \boldsymbol{x}\right)} \tag{35}$$

where $\boldsymbol{w}_m, \forall m$ are weight vectors like those in (32) and (33). This model is also called multinomial logistic regression.

ME minimizes the following regularized negative log-likelihood:

$$\min_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_m} \quad \frac{1}{2}\sum_{m=1}^{k} \|\boldsymbol{w}_k\|^2 + C\sum_{i=1}^{l} \xi_{\mathrm{ME}}\left(\{\boldsymbol{w}_m\}_{m=1}^{k}; \boldsymbol{x}_i, y_i\right) \tag{36}$$

where

$$\xi_{\mathrm{ME}}\left(\{\boldsymbol{w}_m\}_{m=1}^{k}; \boldsymbol{x}, y\right) \equiv -\log P(y|\boldsymbol{x}).$$

Clearly, (36) is similar to (33) and $\xi_{\mathrm{ME}}(\cdot)$ can be considered as a loss function. If $\boldsymbol{w}_{y_i}^T \boldsymbol{x}_i \gg \boldsymbol{w}_m^T \boldsymbol{x}_i, \forall m \neq y_i$, then $\xi_{\mathrm{ME}}(\{\boldsymbol{w}_m\}_{m=1}^{k}; \boldsymbol{x}_i, y_i)$ is close to zero (i.e., no loss). On the other hand, if $\boldsymbol{w}_{y_i}^T \boldsymbol{x}_i$ is smaller than other $\boldsymbol{w}_m^T \boldsymbol{x}_i$, $m \neq y_i$, then $P(y_i|\boldsymbol{x}_i) \ll 1$ and the loss is large. For prediction, the decision function is also (32).

NLP applications often consider a more general ME model by using a function $\boldsymbol{f}(\boldsymbol{x}, y)$ to generate the feature vector

$$P(y|\boldsymbol{x}) \equiv \frac{\exp(\boldsymbol{w}^T \boldsymbol{f}(\boldsymbol{x}, y))}{\sum_{y'} \exp(\boldsymbol{w}^T \boldsymbol{f}(\boldsymbol{x}, y'))}. \tag{37}$$

[6]Details of the connection between logistic regression and maximum entropy can be found in, for example, [12, Sec. 5.2].

Table 3 Comparison of Methods for Multiclass Linear Classification in Storage (Model Size) and Testing Time. n Is the Number of Features and k Is the Number of Classes

Method	Storage	Testing
one-against-rest	$O(kn)$	$O(kn)$
one-against-one	$O(k^2n)$	$O(k^2n)$
DAGSVM	$O(k^2n)$	$O(kn)$
Crammer and Singer's	$O(kn)$	$O(kn)$
maximum entropy	$O(kn)$	$O(kn)$

Equation (35) is a special case of (37) by

$$\boldsymbol{f}(\boldsymbol{x}_i, y) = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \boldsymbol{x}_i \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}0\\ \vdots\\ 0\end{matrix}}\right\} y-1 \\ \\ \\ \\ \\ \end{matrix} \in \mathbf{R}^{nk} \quad \text{and} \quad \boldsymbol{w} = \begin{bmatrix} \boldsymbol{w}_1 \\ \vdots \\ \boldsymbol{w}_k \end{bmatrix}. \tag{38}$$

Many studies have investigated optimization methods for L2-regularized ME. For example, Malouf [66] compares iterative scaling methods [67], gradient descent, nonlinear conjugate gradient, and L-BFGS (quasi-Newton) method [68] to solve (36). Experiments show that quasi-Newton performs better. In [12], a framework is proposed to explain variants of iterative scaling methods [30], [67], [69] and make a connection to coordinate descent methods. For L1-regularized ME, Andrew and Gao [40] propose an extension of L-BFGS.

Recently, instead of solving the primal problem (36), some works solve the dual problem. A detailed derivation of the dual ME is in [33, App. A.7]. Memisevic [70] proposed a two-level decomposition method. Similar to the coordinate descent method [63] for (33) in Section V-B, in [70], a subproblem of k variables is considered at a time. However, the subproblem does not have a closed-form solution, so a second-level coordinate descent method is applied. Collin *et al.* [71] proposed an exponential gradient method to solve ME dual. They also decompose the problem into k-variable subproblems, but only approximately solve each subproblem. The work in [33] follows [70] to apply a two-level coordinate descent method, but uses a different method in the second level to decide variables for update.

D. Comparison

We summarize storage (model size) and testing time of each method in Table 3. Clearly, one-against-one and DAGSVM methods are less practical because of the much higher storage, although the comparison in [57] indicates that one-against-one method gives slightly better testing accuracy. Note that the situation is very different for the kernel case [50], where one-against-one and DAGSVM are very useful methods.

VI. LINEAR-CLASSIFICATION TECHNIQUES FOR NONLINEAR CLASSIFICATION

Many recent developments of linear classification can be extended to handle nonstandard scenarios. Interestingly, most of them are related to training nonlinear classifiers.

A. Training and Testing Explicit Data Mappings via Linear Classifiers

In some problems, training a linear classifier in the original feature space may not lead to competitive performances. For example, on ijcnn1 in Table 1, the testing accuracy (92.21%) of a linear classifier is inferior to 98. 69% of a nonlinear one with the RBF kernel. However, the higher accuracy comes with longer training and testing time. Taking the advantage of linear classifiers' fast training, some studies have proposed using the explicit nonlinear data mappings. That is, we consider $\phi(\boldsymbol{x}_i)$, $i = 1, \ldots, l$, as the new training set and employ a linear classifier. In some problems, this type of approaches may still enjoy fast training/testing, but achieve accuracy close to that of using highly nonlinear kernels.

Some early works, e.g., [72]–[74], have directly trained nonlinearly mapped data in their experiments. Chang *et al.* [9] analyze when this approach leads to faster training and testing. Assume that the coordinate descent method in Section IV-D is used for training linear/kernelized classifiers[7] and $\phi(\boldsymbol{x}) \in \mathbf{R}^d$. From Section IV-D, each coordinate descent step takes $O(d)$ and $O(nl)$ operations for linear and kernelized settings, respectively. Thus, if $d \ll nl$, the approach of training explicit mappings may be faster than using kernels. In [9], the authors particularly study degree-2 polynomial mappings such as (5). The dimensionality is $d = O(n^2)$, but for sparse data, the $O(n^2)$ versus $O(nl)$ comparison is changed to $O(\bar{n}^2)$ versus $O(\bar{n}l)$, where $\bar{n}$ is the average number of nonzero values per instance. For large sparse data sets, $\bar{n} \ll l$, so their approach can be very efficient. Table 4 shows results of training/testing degree-2 polynomial mappings using three data sets in Table 1 with significant lower linear-SVM accuracy than RBF. We apply the same setting as [9, Sec. 4]. From Tables 1 and 4, we observed that training $\phi(\boldsymbol{x}_i), \forall i$ by a linear classifier may give accuracy close to RBF kernel, but is faster in training/testing.

A general framework was proposed in [75] for various nonlinear mappings of data. They noticed that to perform the coordinate descent method in Section IV-D, one only needs that $\boldsymbol{u}^T\phi(\boldsymbol{x})$ in (27) and $\boldsymbol{u} \leftarrow \boldsymbol{u} + y(\alpha_i - \bar{\alpha}_i)\phi(\boldsymbol{x})$ in (28) can be performed. Thus, even if $\phi(\boldsymbol{x})$ cannot be

[7]See the discussion in the end of Section IV-D about the connection between Algorithm 3 and the popular decomposition methods for nonlinear SVMs.

Table 4 Results of Training/Testing Degree-2 Polynomial Mappings by the Coordinate Descent Method in Section IV-D. The Degree-2 Polynomial Mapping Is Dynamically Computed During Training, Instead of Expanded Beforehand. The Last Column Shows the Accuracy Difference Between Degree-2 Polynomial Mappings and RBF SVM

Data set	Degree-2 polynomial			Accuracy difference to RBF kernel
	Time (s)		Testing accuracy	
	Training	Testing		
cod-RNA	1.27	0.12	96.35	-0.3210
ijcnn1	10.38	0.08	97.84	-0.8517
covtype	5,166.30	0.07	80.21	-15.8999

explicitly represented, as long as these two operations can be performed, Algorithm 3 is applicable.

Studies in [76] and [77] designed linear classifiers to train explicit mappings of sequence data, where features correspond to subsequences. Using the relation between subsequences, they are able to design efficient training methods for very high-dimensional mappings.

B. Approximation of Kernel Methods via Linear Classification

Methods in Section VI-A train $\phi(\boldsymbol{x}_i), \forall i$ explicitly, so they obtain the same model as a kernel method using $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$. However, they have limitations when the dimensionality of $\phi(\boldsymbol{x})$ is very high. To resolve the slow training/testing of kernel methods, approximation is sometimes unavoidable. Among the many available methods to approximate the kernel, some of them lead to training a linear classifier. Following [78], we categorize these methods to the following two types.

- *Kernel matrix approximation.* This type of approaches finds a low-rank matrix $\bar{\Phi} \in \mathbf{R}^{d\times l}$ with $d \ll l$ such that $\bar{\Phi}^T\bar{\Phi}$ can approximate the kernel matrix Q

$$\bar{Q} = \bar{\Phi}^T\bar{\Phi} \approx Q. \tag{39}$$

  Assume $\bar{\Phi} \equiv [\bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_l]$. If we replace Q in (16) with $\bar{Q}$, then (16) becomes the dual problem of training a linear SVM on the new set $(y_i, \bar{\boldsymbol{x}}_i)$, $i = 1, \ldots, l$. Thus, optimization methods discussed in Section IV can be directly applied. An advantage of this approach is that we do not need to know an explicit mapping function corresponding to a kernel of our interest (see the other type of approaches discussed below). However, this property causes a complicated testing procedure. That is, the approximation in (39) does not directly reveal how to adjust the decision function (3).

  Early developments focused on finding a good approximation matrix $\bar{\Phi}$. Some examples include Nyström method [79], [80] and incomplete Cholesky factorization [81], [82]. Some works (e.g., [19]) consider approximations other than (39), but also lead to linear classification problems.

A recent study [78] addresses more on training and testing linear SVM after obtaining the low-rank approximation. In particular, details of the testing procedures can be found in [78, Sec. 2.4]. Note that linear SVM problems obtained after kernel approximations are often dense and have more instances than features. Thus, training algorithms suitable for such problems may be different from those for sparse document data.

- *Feature mapping approximation.* This type of approaches finds a mapping function $\bar{\phi} : \mathbf{R}^n \rightarrow \mathbf{R}^d$ such that

$$\bar{\phi}(\boldsymbol{x})^T \bar{\phi}(\boldsymbol{t}) \approx K(\boldsymbol{x}, \boldsymbol{t}).$$

Then, linear classifiers can be applied to new data $\bar{\phi}(\boldsymbol{x}_1), \ldots, \bar{\phi}(\boldsymbol{x}_l)$. The testing phase is straightforward because the mapping $\bar{\phi}(\cdot)$ is available.

Many mappings have been proposed. Examples include random Fourier projection [83], random projections [84], [85], polynomial approximation [86], and hashing [87]–[90]. They differ in various aspects, which are beyond the scope of this paper. An issue related to the subsequent linear classification is that some methods (e.g., [83]) generate dense $\bar{\phi}(\boldsymbol{x})$ vectors, while others give sparse vectors (e.g., [85]). A recent study focusing on the linear classification after obtaining $\bar{\phi}(\boldsymbol{x}_i), \forall i$ is in [91].

VII. TRAINING LARGE DATA BEYOND THE MEMORY OR THE DISK CAPACITY

Recall that we described some binary linear classification algorithms in Section IV. Those algorithms can work well under the assumption that the training set is stored in the computer memory. However, as the training size goes beyond the memory capacity, traditional algorithms may become very slow because of frequent disk access. Indeed, even if the memory is enough, loading data to memory may take more time than subsequent computation [92]. Therefore, the design of algorithms for data larger than memory is very different from that of traditional algorithms.

If the data set is beyond the disk capacity of a single computer, then it must be stored distributively. Internet companies now routinely handle such large data sets in data centers. In such a situation, linear classification faces even more challenges because of expensive communication cost between different computing nodes. In some recent works [93], [94], parallel SVM on distributed environments has been studied but they investigated only kernel SVM. The communication overhead is less serious because of expensive kernel computation. For distributed linear classification, the research is still in its infancy. The current trend is to design algorithms so that computing nodes access data locally and the communication between nodes is minimized. The implementation is often conducted using distributed computing environments such as Hadoop [95]. In this section, we will discuss some ongoing research results.

Among the existing developments, some can be easily categorized as online methods. We describe them in Section VII-A. Batch methods are discussed in Section VII-B, while other approaches are in Section VII-C.

A. Online Methods

An online method updates the model $\boldsymbol{w}$ via using some instances at a time rather than considering the whole training data. Therefore, not only can online methods handle data larger than memory, but also they are suitable for streaming data where each training instance is used only once. One popular online algorithm is the stochastic gradient descent (SGD) method, which can be traced back to stochastic approximation method [96], [97]. Take the primal L2-regularized L1-loss SVM in (7) as an example. At each step, a training instance $\boldsymbol{x}_i$ is chosen and $\boldsymbol{w}$ is updated by

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla^S \left(\frac{1}{2} \|\boldsymbol{w}\|_2^2 + C \max(0, 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i) \right) \qquad (40)$$

where ∇^S is a subgradient operator and η is the learning rate. Specifically, (40) becomes the following update rule:

$$\begin{aligned} &\text{If} \quad 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i > 0, \\ &\text{then} \quad \boldsymbol{w} \leftarrow (1-\eta)\boldsymbol{w} + \eta C y_i \boldsymbol{x}_i. \end{aligned} \qquad (41)$$

The learning rate η is gradually reduced along iterations.

It is well known that SGD methods have slow convergence. However, they are suitable for large data because of accessing only one instance at a time. Early studies which have applied SGD to linear classification include, for example, [98] and [99]. For data with many features, recent studies [5], [26] show that SGD is effective. They allow more flexible settings such as using more than one training instance at a time. We briefly discuss the online setting of Pegasos [5]. In Algorithm 1, at each step a), a small random subset B is used instead of the full set. Similar convergence properties to that described in Section IV-B still hold but in expectation (see [5, Th. 2]).

Instead of solving the primal problem, we can design an online algorithm to solve the dual problem [6], [100]. For example, the coordinate descent method in Algorithm 3 can be easily extended to an online setting by replacing the sequential selection of variables with a random selection. Notice that the update rule (28) is similar to (41), but has the advantage of not needing to decide the learning rate η. This online setting falls into the general framework of randomized coordinate descent methods in [101] and [102]. Using the proof in [101], the linear convergence in expectation is obtained in [6, App. 7.5].

To improve the convergence of SGD, some [103], [104] have proposed using higher order information. The rule in (40) is replaced by

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta H \nabla^S(\cdot) \tag{42}$$

where H is an approximation of the inverse Hessian $\nabla^2 f(\boldsymbol{w})^{-1}$. To save the cost at each update, practically H is a diagonal scaling matrix. Experiments [103] and [104] show that using (42) is faster than (40).

The update rule in (40) assumes L2 regularization. While SGD is applicable for other regularization, it may not perform as well because of not taking special properties of the regularization term into consideration. For example, if L1 regularization is used, a standard SGD may face difficulties to generate a sparse $\boldsymbol{w}$. To address this problem, recently several approaches have been proposed [105]–[110]. The stochastic coordinate descent method in [106] has been extended to a parallel version [111].

Unfortunately, most existing studies of online algorithms conduct experiments by assuming enough memory and reporting the number of times to access data. To apply them in a real scenario without sufficient memory, many practical issues must be checked. Vowpal-Wabbit [112] is one of the very few implementations which can handle data larger than memory. Because the same data may be accessed several times and the disk reading time is expensive, at the first pass, Vowpal-Wabbit stores data to a compressed cache file. This is similar to the compression strategy in [92], which will be discussed in Section VII-B. Currently, Vowpal-Wabbit supports unregularized linear classification and regression. It is extended to solve L1-regularized problems in [105].

Recently, Vowpal-Wabbit (after version 6.0) has supported distributed online learning using the Hadoop [95] framework. We are aware that other Internet companies have constructed online linear classifiers on distributed environments, although details have not been fully available. One example is the system SETI at Google [113].

B. Batch Methods

In some situations, we still would like to consider the whole training set and solve a corresponding optimization problem. While this task is very challenging, some (e.g., [92] and [114]) have checked the situation that data are larger than memory but smaller than disk. Because of expensive disk input/output (I/O), they design algorithms by reading a continuous chunk of data at a time and minimizing the number of disk accesses. The method in [92] extends the coordinate descent method in Section IV-D for linear SVM. The major change is to update more variables at a time so that a block of data is used together. Specifically, in the beginning, the training set is randomly partitioned to m files $B_1, \ldots, B_m$. The available memory space needs to be able to accommodate one block of data and the working space of a training algorithm. To solve (16), sequentially one block of data B is read and the following function of $\boldsymbol{d}$ is minimized under the condition $0 \leq \alpha_i + d_i \leq C, \forall i \in B$ and $d_i = 0, \forall i \notin B$

$$\begin{aligned} f^D(\boldsymbol{\alpha} + \boldsymbol{d}) - f^D(\boldsymbol{\alpha}) &= \frac{1}{2}\boldsymbol{d}_B^T Q_{BB} \boldsymbol{d}_B + \boldsymbol{d}_B^T (Q\boldsymbol{\alpha} - \boldsymbol{e})_B \\ &= \frac{1}{2}\boldsymbol{d}_B^T Q_{BB} \boldsymbol{d}_B + \sum_{i \in B} y_i d_i (\boldsymbol{u}^T \boldsymbol{x}_i) - \boldsymbol{d}_B^T \boldsymbol{e}_B \end{aligned} \tag{43}$$

where Q_{BB} is a submatrix of Q and $\boldsymbol{u}$ is defined in (26). By maintaining $\boldsymbol{u}$ in a way similar to (28), equation (43) involves only data in the block B, which can be stored in memory. Equation (43) can be minimized by any traditional algorithm. Experiments in [92] demonstrate that they can train data 20 times larger than the memory capacity. This method is extended in [115] to cache informative data points in the computer memory. That is, at each iteration, not only the selected block but also the cached points are used for updating corresponding variables. Their way to select informative points is inspired by the shrinking techniques used in training nonlinear SVM [8], [47].

For distributed batch learning, all existing parallel optimization methods [116] can possibly be applied. However, we have not seen many practical deployments for training large-scale data. Recently, Boyd *et al.* [117] have considered the alternating direction method of multiplier (ADMM) [118] for distributed learning. Take SVM as an example and assume data points are partitioned to m distributively stored sets $B_1, \ldots, B_m$. This method solves the following approximation of the original optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_m,\boldsymbol{z}} \quad & \frac{1}{2}\boldsymbol{z}^T\boldsymbol{z} + C\sum_{j=1}^{m}\sum_{i \in B_j} \xi_{L1}(\boldsymbol{w}_j; \boldsymbol{x}_i, y_i) \\ & + \frac{\rho}{2}\sum_{j=1}^{m} \|\boldsymbol{w}_j - \boldsymbol{z}\|^2 \\ \text{subject to} \quad & \boldsymbol{w}_j - \boldsymbol{z} = \boldsymbol{0}, \forall j \end{aligned}$$

where ρ is a prespecific parameter. It then employs an optimization method of multipliers by alternatively minimizing the Lagrangian function over $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m$, minimizing the Lagrangian over $\boldsymbol{z}$, and updating dual multipliers. The minimization of Lagrangian over $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m$ can be decomposed to m independent problems. Other steps do not involve data at all. Therefore, data points are locally accessed and the communication cost is kept minimum. Examples of using ADMM for distributed training include [119]. Some known problems of this approaches are first that the convergence rate is not very fast, and second that it is unclear how to choose parameter ρ.

Some works solve an optimization problem using parallel SGD. The data are stored in a distributed system, and each node only computes the subgradient corresponding to the data instances in the node. In [120], a delayed SGD is proposed. Instead of computing the subgradient of the current iterate $\boldsymbol{w}^k$, in delayed SGD, each node computes the subgradient of a previous iterator $\boldsymbol{w}^{\tau(k)}$, where $\tau(k) \leq k$. Delayed SGD is useful to reduce the synchronization delay because of communication overheads or uneven computational time at various nodes. Recent works [121], [122] show that delayed SGD is efficient when the number of nodes is large, and the delay is asymptotically negligible.

C. Other Approaches

We briefly discuss some other approaches which cannot be clearly categorized as batch or online methods.

The most straightforward method to handle large data is probably to randomly select a subset that can fit in memory. This approach works well if the data quality is good; however, sometimes using more data gives higher accuracy. To improve the performance of using only a subset, some have proposed techniques to include important data points into the subset. For example, the approach in [123] selects a subset by reading data from disk only once. For data in a distributed environment, subsampling can be a complicated operation. Moreover, a subset fitting the memory of one single computer may be too small to give good accuracy.

Bagging [124] is a popular classification method to split a learning task to several easier ones. It selects several random subsets, trains each of them, and ensembles (e.g., averaging) the results during testing. This method may be particularly useful for distributively stored data because we can directly consider data in each node as a subset. However, if data quality in each node is not good (e.g., all instances with the same class label), the model generated by each node may be poor. Thus, ensuring data quality of each subset is a concern. Some studies have applied the bagging approach on a distributed system [125], [126]. For example, in the application of web advertising, Chakrabarti *et al.* [125] train a set of individual classifiers in a distributed way. Then, a final model is obtained by averaging the separate classifiers. In the linguistic applications, McDonald *et al.* [127] extend the simple model average to the weighted average and achieve better performance. An advantage of the bagging-like approach is the easy implementation using distributed computing techniques such as MapReduce [128].[8]

VIII. RELATED TOPICS

In this section, we discuss some other linear models. They are related to linear classification models discussed in earlier sections.

A. Structured Learning

In the discussion so far, we assumed that the label y_i is a single value. For binary classification, it is $+1$ or -1, while for multiclass classification, it is one of the k class labels. However, in some applications, the label may be a more sophisticated object. For example, in part-of-speech (POS) tagging applications, the training instances are sentences and the labels are sequences of POS tags of words. If there are l sentences, we can write the training instances as $(\boldsymbol{y}_i, \boldsymbol{x}_i) \in Y^{n_i} \times X^{n_i}, \forall i = 1, \ldots, l$, where $\boldsymbol{x}_i$ is the ith sentence, $\boldsymbol{y}_i$ is a sequence of tags, X is a set of unique words in the context, Y is a set of candidate tags for each word, and n_i is the number of words in the ith sentence. Note that we may not be able to split the problem to several independent ones by treating each value y_{ij} of $\boldsymbol{y}_i$ as the label, because y_{ij} not only depends on the sentence $\boldsymbol{x}_i$ but also other tags $(y_{i1}, \ldots, y_{i(j-1)}, y_{i(j+1)}, \ldots y_{in_i})$. To handle these problems, we could use structured learning models like conditional random fields [65] and structured SVM [129], [130].

- *Conditional random fields (CRFs)*. The CRF [65] is a linear structured model commonly used in NLP. Using notation mentioned above and a feature function $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$ like ME, CRF solves the following problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{l} \xi_{\text{CRF}}(\boldsymbol{w}; \boldsymbol{x}_i, \boldsymbol{y}_i) \tag{44}$$

where

$$\xi_{\text{CRF}}(\boldsymbol{w}; \boldsymbol{x}_i, \boldsymbol{y}_i) \equiv -\log P(\boldsymbol{y}_i|\boldsymbol{x}_i)$$
$$P(\boldsymbol{y}|\boldsymbol{x}) \equiv \frac{\exp(\boldsymbol{w}^T\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}))}{\sum_{\boldsymbol{y}'} \exp(\boldsymbol{w}^T\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}'))}. \tag{45}$$

If elements in $\boldsymbol{y}_i$ are independent of each other, then CRF reduces to ME.

[8]We mentioned earlier the Hadoop system, which includes a MapReduce implementation.

The optimization of (44) is challenging because in the probability model (45), the number of possible $\boldsymbol{y}$'s is exponentially large. An important property to make CRF practical is that the gradient of the objective function in (44) can be efficiently evaluated by dynamic programming [65]. Some available optimization methods include L-BFGS (quasi-Newton) and conjugate gradient [131], SGD [132], stochastic quasi-Newton [103], [133], and trust region Newton method [134]. It is shown in [134] that the Hessian-vector product (23) of the Newton method can also be evaluated by dynamic programming.

- *Structured SVM*. Structured SVM solves the following optimization problem generalized form multiclass SVM in [59], [60]:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{l} \xi_{\rm SS}(\boldsymbol{w};\boldsymbol{x}_i,\boldsymbol{y}_i) \tag{46}$$

where

$$\xi_{\rm SS}(\boldsymbol{w};\boldsymbol{x}_i,\boldsymbol{y}_i) \equiv \max_{\boldsymbol{y}\neq\boldsymbol{y}_i} \Big(\max \big(0, \Delta(\boldsymbol{y}_i,\boldsymbol{y}) - \boldsymbol{w}^T(\boldsymbol{f}(\boldsymbol{x}_i,\boldsymbol{y}_i) - \boldsymbol{f}(\boldsymbol{x}_i,\boldsymbol{y}))\big)\Big)$$

and $\Delta(\cdot)$ is a distance function with $\Delta(\boldsymbol{y}_i,\boldsymbol{y}_i)=0$ and $\Delta(\boldsymbol{y}_i,\boldsymbol{y}_j)=\Delta(\boldsymbol{y}_j,\boldsymbol{y}_i)$. Similar to the relation between conditional random fields and maximum entropy, if

$$\Delta(\boldsymbol{y}_i,\boldsymbol{y}_j) = \begin{cases} 0, & \text{if } \boldsymbol{y}_i = \boldsymbol{y}_j \\ 1, & \text{otherwise} \end{cases}$$

and $\boldsymbol{y}_i \in \{1,\ldots,k\}, \forall i$, then structured SVM becomes Crammer and Singer's problem in (33) following the definition of $\boldsymbol{f}(\boldsymbol{x},y)$ and $\boldsymbol{w}$ in (38).

Like CRF, the main difficulty to solve (46) is on handling an exponential number of $\boldsymbol{y}$ values. Some works (e.g., [25], [129], and [135]) use a cutting plane method [136] to solve (46). In [137], a stochastic subgradient descent method is applied for both online and batch settings.

B. Regression

Given training data $\{(z_i,\boldsymbol{x}_i)\}_{i=1}^{l} \subset \mathbf{R}\times\mathbf{R}^n$, a regression problem finds a weight vector $\boldsymbol{w}$ such that $\boldsymbol{w}^T\boldsymbol{x}_i \approx z_i, \forall i$. Like classification, a regression task solves a risk minimization problem involving regularization and loss terms. While L1 and L2 regularization is still used, loss functions are different, where two popular ones are

$$\xi_{\rm LS}(\boldsymbol{w};\boldsymbol{x},z) \equiv \frac{1}{2}(z-\boldsymbol{w}^T\boldsymbol{x})^2 \tag{47}$$

$$\xi_{\epsilon}(\boldsymbol{w};\boldsymbol{x},z) \equiv \max\big(0, |z-\boldsymbol{w}^T\boldsymbol{x}|-\epsilon\big). \tag{48}$$

The least square loss in (47) is widely used in many places, while the ϵ-insensitive loss in (48) is extended from the L1 loss in (8), where there is a user-specified parameter ϵ as the error tolerance. Problem (7) with L2 regularization and ϵ-insensitive loss is called support vector regression (SVR) [138]. Contrary to the success of linear classification, so far not many applications of linear regression on large sparse data have been reported. We believe that this topic has not been fully explored yet.

Regarding the minimization of (7), if L2 regularization is used, many optimization methods mentioned in Section IV can be easily modified for linear regression.

We then particularly discuss L1-regularized least square regression, which has recently drawn much attention for signal processing and image applications. This research area is so active that many optimization methods (e.g., [49] and [139]–[143]) have been proposed. However, as pointed out in [13], optimization methods most suitable for signal/image applications via L1-regularized regression may be very different from those in Section IV for classifying large sparse data. One reason is that data from signal/image problems tend to be dense. Another is that $\boldsymbol{x}_i, \forall i$ may be not directly available in some signal/image problems. Instead, we can only evaluate the product between the data matrix and a vector through certain operators. Thus, optimization methods that can take this property into their design may be more efficient.

IX. CONCLUSION

In this paper, we have comprehensively reviewed recent advances of large linear classification. For some applications, linear classifiers can give comparable accuracy to nonlinear classifiers, but enjoy much faster training and testing speed. However, these results do not imply that nonlinear classifiers should no longer be considered. Both linear and nonlinear classifiers are useful under different circumstances.

Without mapping data to another space, for linear classification we can easily prepare, select, and manipulate features. We have clearly shown that linear classification is not limited to standard scenarios like document classification. It can be applied in many other places such as efficiently approximating nonlinear classifiers. We are confident that future research works will make linear classification a useful technique for more large-scale applications. ■

REFERENCES

[1] B. E. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," in *Proc. 5th Annu. Workshop Comput. Learn. Theory*, 1992, pp. 144–152.

[2] C. Cortes and V. Vapnik, "Support-vector network," *Mach. Learn.*, vol. 20, pp. 273–297, 1995.

[3] J. S. Cramer, "The origins of logistic regression," Tinbergen Inst., Amsterdam, The Netherlands, Tech. Rep. [Online]. Available: http://ideas.repec.org/p/dgr/uvatin/20020119.html

[4] T. Joachims, "Training linear SVMs in linear time," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2006, DOI: 10.1145/1150402.1150429.

[5] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Pegasos: Primal estimated sub-gradient solver for SVM," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, DOI: 10.1145/1273496.1273598.

[6] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan, "A dual coordinate descent method for large-scale linear SVM," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, DOI: 10.1145/1390156.1390208. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/cddual.pdf

[7] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. (2008). LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.* [Online]. *9*, pp. 1871–1874. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf

[8] C.-C. Chang and C.-J. Lin. (2011). LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* [Online]. *2*, pp. 27:1–27:27. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm

[9] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin. (2010). Training and testing low-degree polynomial data mappings via linear SVM. *J. Mach. Learn. Res.* [Online]. *11*, pp. 1471–1490. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/lowpoly_journal.pdf

[10] S. S. Keerthi and C.-J. Lin, "Asymptotic behaviors of support vector machines with Gaussian kernel," *Neural Comput.*, vol. 15, no. 7, pp. 1667–1689, 2003.

[11] Z. S. Harris, "Distributional structure," *Word*, vol. 10, pp. 146–162, 1954.

[12] F.-L. Huang, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin. (2010). Iterative scaling and coordinate descent methods for maximum entropy. *J. Mach. Learn. Res.* [Online]. *11*, pp. 815–848. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/maxent_journal.pdf

[13] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. (2010). A comparison of optimization methods and software for large-scale L_1-regularized linear classification. *J. Mach. Learn. Res.* [Online]. *11*, pp. 3183–3234. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/l1.pdf

[14] O. L. Mangasarian, "A finite Newton method for classification," *Optim. Methods Softw.*, vol. 17, no. 5, pp. 913–929, 2002.

[15] A. Y. Ng, "Feature selection, L_1 vs. L_2 regularization, and rotational invariance," in *Proc. 21st Int. Conf. Mach. Learn.*, 2004, DOI: 10.1145/1015330.1015435.

[16] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Stat. Soc. B*, vol. 58, pp. 267–288, 1996.

[17] D. L. Donoho and Y. Tsaig, "Fast solution of ℓ_1-norm minimization problems when the solution may be sparse," *IEEE Trans. Inf. Theory*, vol. 54, no. 11, pp. 4789–4812, Nov. 2008.

[18] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *J. Roy. Stat. Soc. B (Stat. Methodol.)*, vol. 67, no. 2, pp. 301–320, 2005.

[19] Y.-J. Lee and O. L. Mangasarian, "RSVM: Reduced support vector machines," in *Proc. 1st SIAM Int. Conf. Data Mining*, 2001. [Online]. Available: http://www.siam.org/proceedings/datamining/2001/dm01.php

[20] J. Shi, W. Yin, S. Osher, and P. Sajda, "A fast hybrid algorithm for large scale ℓ_1-regularized logistic regression," *J. Mach. Learn. Res.*, vol. 11, pp. 713–741, 2010.

[21] K.-W. Chang, C.-J. Hsieh, and C.-J. Lin. (2008). Coordinate descent method for large-scale L2-loss linear SVM. *J. Mach. Learn. Res.* [Online]. *9*, pp. 1369–1398. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/cdl2.pdf

[22] G.-X. Yuan, C.-H. Ho, and C.-J. Lin. (2011). An improved GLMNET for ℓ_1-regularized logistic regression and support vector machines, Nat. Taiwan Univ., Taipei, Taiwan, Tech. Rep. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/long-glmnet.pdf

[23] P. S. Bradley and O. L. Mangasarian, "Massive data discrimination via linear support vector machines," *Optim. Methods Softw.*, vol. 13, no. 1, pp. 1–10, 2000.

[24] V. Franc and S. Sonnenburg, "Optimized cutting plane algorithm for support vector machines," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 320–327.

[25] C. H. Teo, S. Vishwanathan, A. Smola, and Q. V. Le, "Bundle methods for regularized risk minimization," *J. Mach. Learn. Res.*, vol. 11, pp. 311–365, 2010.

[26] L. Bottou, *Stochastic Gradient Descent Examples*, 2007. [Online]. Available: http://leon.bottou.org/projects/sgd

[27] S. S. Keerthi and D. DeCoste, "A modified finite Newton method for fast solution of large scale linear SVMs," *J. Mach. Learn. Res.*, vol. 6, pp. 341–361, 2005.

[28] C.-J. Lin, R. C. Weng, and S. S. Keerthi. (2008). Trust region Newton method for large-scale logistic regression. *J. Mach. Learn. Res.* [Online]. *9*, pp. 627–650. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/logistic.pdf

[29] T. P. Minka, *A Comparison of Numerical Optimizers for Logistic Regression*, 2003. [Online]. Available: http://research.microsoft.com/~minka/papers/logreg/

[30] J. Goodman, "Sequential conditional generalized iterative scaling," in *Proc. 40th Annu. Meeting Assoc. Comput. Linguist.*, 2002, pp. 9–16.

[31] R. Jin, R. Yan, J. Zhang, and A. G. Hauptmann, "A faster iterative scaling algorithm for conditional exponential model," in *Proc. 20th Int. Conf. Mach. Learn.*, 2003, pp. 282–289.

[32] P. Komarek and A. W. Moore, "Making logistic regression a core data mining tool: A practical investigation of accuracy, speed, and simplicity," Robotics Inst., Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep. TR-05-27, 2005.

[33] H.-F. Yu, F.-L. Huang, and C.-J. Lin, "Dual coordinate descent methods for logistic regression and maximum entropy models," *Mach. Learn.*, vol. 85, no. 1–2, pp. 41–75, Oct. 2011. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/maxent_dual.pdf

[34] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani, "1-norm support vector machines," in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. Cambridge, MA: MIT Press, 2004.

[35] G. M. Fung and O. L. Mangasarian, "A feature selection Newton method for support vector machine classification," *Comput. Optim. Appl.*, vol. 28, pp. 185–202, 2004.

[36] O. L. Mangasarian, "Exact 1-norm support vector machines via unconstrained convex differentiable minimization," *J. Mach. Learn. Res.*, vol. 7, pp. 1517–1530, 2006.

[37] K. Koh, S.-J. Kim, and S. Boyd. (2007). An interior-point method for large-scale L_1-regularized logistic regression. *J. Mach. Learn. Res.* [Online]. *8*, pp. 1519–1555. Available: http://www.stanford.edu/~boyd/l1_logistic_reg.html

[38] A. Genkin, D. D. Lewis, and D. Madigan, "Large-scale Bayesian logistic regression for text categorization," *Technometrics*, vol. 49, no. 3, pp. 291–304, 2007.

[39] S. Yun and K.-C. Toh, "A coordinate gradient descent method for L_1-regularized convex minimization," *Comput. Optim. Appl.*, vol. 48, no. 2, pp. 273–307, 2011.

[40] G. Andrew and J. Gao, "Scalable training of L_1-regularized log-linear models," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, DOI: 10.1145/1273496.1273501.

[41] J. H. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *J. Stat. Softw.*, vol. 33, no. 1, pp. 1–22, 2010.

[42] J. Liu, J. Chen, and J. Ye, "Large-scale sparse logistic regression," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2009, pp. 547–556.

[43] R. Tomioka, T. Suzuki, and M. Sugiyama, "Super-linear convergence of dual augmented Lagrangian algorithm for sparse learning," *J. Mach. Learn. Res.*, vol. 12, pp. 1537–1586, 2011.

[44] M. Schmidt, G. Fung, and R. Rosales, Optimization methods for L_1-regularization, Univ. British Columbia, Vancouver, BC, Canada, Tech. Rep. TR-2009-19, 2009.

[45] C.-J. Lin and J. J. Moré, "Newton's method for large-scale bound constrained problems," *SIAM J. Optim.*, vol. 9, pp. 1100–1127, 1999.

[46] Z.-Q. Luo and P. Tseng, "On the convergence of coordinate descent method for convex differentiable minimization," *J. Optim. Theory Appl.*, vol. 72, no. 1, pp. 7–35, 1992.

[47] T. Joachims, "Making large-scale SVM learning practical," in *Advances in Kernel Methods—Support Vector Learning*, B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds. Cambridge, MA: MIT Press, 1998, pp. 169–184.

[48] J. H. Friedman, T. Hastie, H. Höfling, and R. Tibshirani, "Pathwise coordinate optimization," *Ann. Appl. Stat.*, vol. 1, no. 2, pp. 302–332, 2007.

[49] S. J. Wright, R. D. Nowak, and M. A. Figueiredo, "Sparse reconstruction by separable approximation," *IEEE Trans. Signal Process.*, vol. 57, no. 7, pp. 2479–2493, Jul. 2009.

[50] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multi-class support vector machines," *IEEE Trans. Neural Netw.*, vol. 13, no. 2, pp. 415–425, Mar. 2002.

[51] R. Rifkin and A. Klautau, "In defense of one-vs-all classification," *J. Mach. Learn. Res.*, vol. 5, pp. 101–141, 2004.

[52] E. L. Allwein, R. E. Schapire, and Y. Singer, "Reducing multiclass to binary: A unifying approach for margin classifiers," *J. Mach. Learn. Res.*, vol. 1, pp. 113–141, 2001.

[53] T.-K. Huang, R. C. Weng, and C.-J. Lin. (2006). Generalized Bradley-Terry models and multi-class probability estimates. *J. Mach. Learn. Res.* [Online]. *7*, pp. 85–115. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/generalBT.pdf

[54] L. Bottou, C. Cortes, J. Denker, H. Drucker, I. Guyon, L. Jackel, Y. LeCun, U. Muller, E. Sackinger, P. Simard, and V. Vapnik, "Comparison of classifier methods: A case study in handwriting digit recognition," in *Proc. Int. Conf. Pattern Recognit.*, 1994, pp. 77–87.

[55] S. Knerr, L. Personnaz, and G. Dreyfus, "Single-layer learning revisited: A stepwise procedure for building and training a neural network," in *Neurocomputing: Algorithms, Architectures and Applications*, J. Fogelman, Ed. New York: Springer-Verlag, 1990.

[56] J. H. Friedman, "Another approach to polychotomous classification," Dept. Stat., Stanford Univ., Stanford, CA, Tech. Rep. [Online]. Available: http://www-stat.stanford.edu/~jhf/ftp/poly.pdf

[57] T.-L. Huang, "Comparison of L2-regularized multi-class linear classifiers," M.S. thesis, Dept. Comput. Sci. Inf. Eng., Nat. Taiwan Univ., Taipei, Taiwan, 2010.

[58] J. C. Platt, N. Cristianini, and J. Shawe-Taylor, "Large margin DAGs for multiclass classification," in *Advances in Neural Information Processing Systems*, vol. 12. Cambridge, MA: MIT Press, 2000, pp. 547–553.

[59] J. Weston and C. Watkins, "Multi-class support vector machines," in *Proc. Eur. Symp. Artif. Neural Netw.*, M. Verleysen, Ed., Brussels, 1999, pp. 219–224.

[60] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *J. Mach. Learn. Res.*, vol. 2, pp. 265–292, 2001.

[61] Y. Lee, Y. Lin, and G. Wahba, "Multicategory support vector machines," *J. Amer. Stat. Assoc.*, vol. 99, no. 465, pp. 67–81, 2004.

[62] C.-J. Lin. (2002, Sep.). A formal analysis of stopping criteria of decomposition methods for support vector machines. *IEEE Trans. Neural Netw.* [Online]. *13(5)*, pp. 1045–1052. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/stop.ps.gz

[63] S. S. Keerthi, S. Sundararajan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, "A sequential dual method for large scale multi-class linear SVMs," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2008, pp. 408–416. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/sdm_kdd.pdf

[64] A. L. Berger, V. J. Della Pietra, and S. A. Della Pietra, "A maximum entropy approach to natural language processing," *Comput. Linguist.*, vol. 22, no. 1, pp. 39–71, 1996.

[65] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proc. 18th Int. Conf. Mach. Learn.*, 2001, pp. 282–289.

[66] R. Malouf, "A comparison of algorithms for maximum entropy parameter estimation," in *Proc. 6th Conf. Natural Lang. Learn.*, 2002, DOI: 10.3115/1118853.1118871.

[67] J. N. Darroch and D. Ratcliff, "Generalized iterative scaling for log-linear models," *Ann. Math. Stat.*, vol. 43, no. 5, pp. 1470–1480, 1972.

[68] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Math. Programm.*, vol. 45, no. 1, pp. 503–528, 1989.

[69] S. Della Pietra, V. Della Pietra, and J. Lafferty, "Inducing features of random fields," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 4, pp. 380–393, Apr. 1997.

[70] R. Memisevic, "Dual optimization of conditional probability models," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2006.

[71] M. Collins, A. Globerson, T. Koo, X. Carreras, and P. Bartlett, "Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks," *J. Mach. Learn. Res.*, vol. 9, pp. 1775–1822, 2008.

[72] E. M. Gertz and J. D. Griffin, "Support vector machine classifiers for large data sets," Argonne Nat. Lab., Argonne, IL, Tech. Rep. ANL/MCS-TM-289, 2005.

[73] J. H. Jung, D. P. O'Leary, and A. L. Tits, "Adaptive constraint reduction for training support vector machines," *Electron. Trans. Numer. Anal.*, vol. 31, pp. 156–177, 2008.

[74] Y. Moh and J. M. Buhmann, "Kernel expansion for online preference tracking," in *Proc. Int. Soc. Music Inf. Retrieval*, 2008, pp. 167–172.

[75] S. Sonnenburg and V. Franc, "COFFIN: A computational framework for linear SVMs," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010, pp. 999–1006.

[76] G. Ifrim, G. Bakır, and G. Weikum, "Fast logistic regression for text categorization with variable-length n-grams," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2008, pp. 354–362.

[77] G. Ifrim and C. Wiuf, "Bounded coordinate-descent for biological sequence classification in high dimensional predictor space," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2011, DOI: 10.1145/2020408.2020519.

[78] S. Lee and S. J. Wright, "ASSET: Approximate stochastic subgradient estimation training for support vector machines," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2012.

[79] C. K. I. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," in *Advances in Neural Information Processing Systems 13*, T. Leen, T. Dietterich, and V. Tresp, Eds. Cambridge, MA: MIT Press, 2001, pp. 682–688.

[80] P. Drineas and M. W. Mahoney, "On the Nyström method for approximating a gram matrix for improved kernel-based learning," *J. Mach. Learn. Res.*, vol. 6, pp. 2153–2175, 2005.

[81] S. Fine and K. Scheinberg, "Efficient SVM training using low-rank kernel representations," *J. Mach. Learn. Res.*, vol. 2, pp. 243–264, 2001.

[82] F. R. Bach and M. I. Jordan, "Predictive low-rank decomposition for kernel methods," in *Proc. 22nd Int. Conf. Mach. Learn.*, 2005, pp. 33–40.

[83] A. Rahimi and B. Recht, "Random features for large-scale kernel machines *Advances in Neural Information Processing Systems*. Cambridge, MA: MIT Press, 2008, pp. 1177–1184.

[84] D. Achlioptas, "Database-friendly random projections: Johnson-Lindenstrauss with binary coins," *J. Comput. Syst. Sci.*, vol. 66, pp. 671–687, 2003.

[85] P. Li, T. J. Hastie, and K. W. Church, "Very sparse random projections," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2006, pp. 287–296.

[86] K.-P. Lin and M.-S. Chen, "Efficient kernel approximation for large-scale support vector machine classification," in *Proc. 11th SIAM Int. Conf. Data Mining*, 2011, pp. 211–222.

[87] Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, A. Strehl, and S. Vishwanathan, "Hash kernels," in *Proc. 12th Int. Conf. Artif. Intell. Stat.*, 2009, vol. 5, pp. 496–503.

[88] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proc. 26th Int. Conf. Mach. Learn.*, 2009, pp. 1113–1120.

[89] P. Li and A. C. König, "b-bit minwise hashing," in *Proc. 19th Int. Conf. World Wide Web*, 2010, pp. 671–680.

[90] P. Li and A. C. König, "Theory and applications of b-bit minwise hashing," *Commun. ACM*, vol. 54, no. 8, pp. 101–109, 2011.

[91] P. Li, A. Shrivastava, J. Moore, and A. C. König, "Hashing algorithms for large-scale learning," Cornell Univ., Ithaca, NY, Tech. Rep. [Online]. Available: http://www.stat.cornell.edu/~li/reports/HashLearning.pdf

[92] H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin, "Large linear classification when data cannot fit in memory," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2010, pp. 833–842. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/kdd_disk_decomposition.pdf

[93] E. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui, "Parallelizing support vector machines on distributed computers," in *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. Cambridge, MA: MIT Press, 2008, pp. 257–264.

[94] Z. A. Zhu, W. Chen, G. Wang, C. Zhu, and Z. Chen, "P-packSVM: Parallel primal gradient descent kernel SVM," in *Proc. IEEE Int. Conf. Data Mining*, 2009, pp. 677–686.

[95] T. White, *Hadoop: The Definitive Guide*, 2nd ed. New York: O'Reilly Media, 2010.

[96] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Stat.*, vol. 22, no. 3, pp. 400–407, 1951.

[97] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *Ann. Math. Stat.*, vol. 23, no. 3, pp. 462–466, 1952.

[98] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Proc. 21st Int. Conf. Mach. Learn.*, 2004, DOI: 10.1145/1015330.1015332.

[99] L. Bottou and Y. LeCun, "Large scale online learning," *Advances in Neural Information Processing Systems 16*. Cambridge, MA: MIT Press, 2004, pp. 217–224.

[100] A. Bordes, S. Ertekin, J. Weston, and L. Bottou, "Fast kernel classifiers with online and active learning," *J. Mach. Learn. Res.*, vol. 6, pp. 1579–1619, 2005.

[101] Y. E. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," Université Catholique de

Louvain, Louvain-la-Neuve, Louvain, Belgium, CORE Discussion Paper, Tech. Rep. [Online]. Available: http://www.ucl.be/cps/ucl/doc/core/documents/coredp2010_2web.pdf

[102] P. Richtárik and M. Takáč, "Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function," Schl. Math., Univ. Edinburgh, Edinburgh, U.K., Tech. Rep., 2011.

[103] A. Bordes, L. Bottou, and P. Gallinari, "SGD-QN: Careful quasi-Newton stochastic gradient descent," *J. Mach. Learn. Res.*, vol. 10, pp. 1737–1754, 2009.

[104] A. Bordes, L. Bottou, P. Gallinari, J. Chang, and S. A. Smith, "Erratum: SGD-QN is less careful than expected," *J. Mach. Learn. Res.*, vol. 11, pp. 2229–2240, 2010.

[105] J. Langford, L. Li, and T. Zhang, "Sparse online learning via truncated gradient," *J. Mach. Learn. Res.*, vol. 10, pp. 771–801, 2009.

[106] S. Shalev-Shwartz and A. Tewari, "Stochastic methods for L_1-regularized loss minimization," *J. Mach. Learn. Res.*, vol. 12, pp. 1865–1892, 2011.

[107] Y. E. Nesterov, "Primal-dual subgradient methods for convex problems," *Math. Programm.*, vol. 120, no. 1, pp. 221–259, 2009.

[108] J. Duchi and Y. Singer, "Efficient online and batch learning using forward backward splitting," *J. Mach. Learn. Res.*, vol. 10, pp. 2899–2934, 2009.

[109] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[110] L. Xiao, "Dual averaging methods for regularized stochastic learning and online optimization," *J. Mach. Learn. Res.*, vol. 11, pp. 2543–2596, 2010.

[111] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin, "Parallel coordinate descent for L_1-regularized loss minimization," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 321–328.

[112] J. Langford, L. Li, and A. Strehl, *Vowpal Wabbit*, 2007. [Online]. Available: https://github.com/JohnLangford/vowpal_wabbit/wiki

[113] S. Tong, *Lessons Learned Developing a Practical Large Scale Machine Learning System*, Google Research Blog, 2010. [Online]. Available: http://googleresearch.blogspot.com/2010/04/lessons-learned-developing-practical.html

[114] M. Ferris and T. Munson, "Interior point methods for massive support vector machines," *SIAM J. Optim.*, vol. 13, no. 3, pp. 783–804, 2003.

[115] K.-W. Chang and D. Roth, "Selective block minimization for faster convergence of limited memory large-scale linear models," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2011, DOI: 10.1145/2020408.2020517.

[116] Y. Censor and S. A. Zenios, *Parallel Optimization: Theory, Algorithms, and Applications*. Oxford, U.K.: Oxford Univ. Press, 1998.

[117] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.

[118] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite element approximation," *Comput. Math. Appl.*, vol. 2, pp. 17–40, 1976.

[119] P. A. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed support vector machines," *J. Mach. Learn.*, vol. 11, pp. 1663–1707, 2010.

[120] A. Nedić, D. P. Bertsekas, and V. S. Borkar, "Distributed asynchronous incremental subgradient methods," *Studies Comput. Math.*, vol. 8, pp. 381–407, 2001.

[121] J. Langford, A. Smola, and M. Zinkevich, "Slow learners are fast," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Cambridge, MA: MIT Press, 2009, pp. 2331–2339.

[122] A. Agarwal and J. Duchi, "Distributed delayed stochastic optimization," in *Advances in Neural Information Processing Systems 24*. Cambridge, MA: MIT Press, 2011.

[123] H. Yu, J. Yang, and J. Han, "Classifying large data sets using SVMs with hierarchical clusters," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Disc. Data Mining*, 2003, pp. 306–315.

[124] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996.

[125] D. Chakrabarti, D. Agarwal, and V. Josifovski, "Contextual advertising by combining relevance with click feedback," in *Proc. 17th Int. Conf. World Wide Web*, 2008, pp. 417–426.

[126] M. Zinkevich, M. Weimer, A. Smola, and L. Li, "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems 23*, J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds. Cambridge, MA: MIT Press, 2010, pp. 2595–2603.

[127] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *Proc. 48th Annu. Meeting Assoc. Comput. Linguist.*, 2010, pp. 456–464.

[128] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[129] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large margin methods for structured and interdependent output variables," *J. Mach. Learn. Res.*, vol. 6, pp. 1453–1484, 2005.

[130] B. Taskar, C. Guestrin, and D. Koller, "Max-margin markov networks," in *Advances in Neural Information Processing Systems 16*. Cambridge, MA: MIT Press, 2004.

[131] F. Sha and F. C. N. Pereira, "Shallow parsing with conditional random fields," in *Proc. HLT-NAACL*, 2003, pp. 134–141.

[132] S. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. Murphy, "Accelerated training of conditional random fields with stochastic gradient methods," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 969–976.

[133] N. N. Schraudolph, J. Yu, and S. Gunter, "A stochastic quasi-Newton method for online convex optimization," in *Proc. 11th Int. Conf. Artif. Intell. Stat.*, 2007, pp. 433–440.

[134] P.-J. Chen, "Newton methods for conditional random fields," M.S. thesis, Dept. Comput. Sci. Inf. Eng., National Taiwan University, Taipei, Taiwan, 2009.

[135] T. Joachims, T. Finley, and C.-N. J. Yu, "Cutting-plane training of structural SVMs," *J. Mach. Learn.*, vol. 77, no. 1, 2008, DOI: 10.1007/s10994-009-5108-8.

[136] J. E. Kelley, "The cutting-plane method for solving convex programs," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 4, pp. 703–712, 1960.

[137] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "(Online) subgradient methods for structured prediction," in *Proc. 11th Int. Conf. Artif. Intell. Stat.*, 2007, pp. 380–387.

[138] V. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[139] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Commun. Pure Appl. Math.*, vol. 57, pp. 1413–1457, 2004.

[140] M. A. T. Figueiredo, R. Nowak, and S. Wright, "Gradient projection for sparse reconstruction: Applications to compressed sensing and other inverse problems," *IEEE J. Sel. Top. Signal Process.*, vol. 1, no. 4, pp. 586–598, Dec. 2007.

[141] S.-J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky, "An interior point method for large-scale L_1-regularized least squares," *IEEE J. Sel. Top. Signal Process.*, vol. 1, no. 4, pp. 606–617, Dec. 2007.

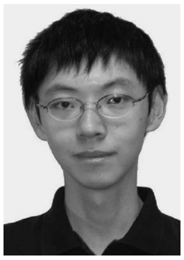
[142] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra, "Efficient projections onto the L_1-ball for learning in high dimensions," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, DOI: 10.1145/1390156.1390191.

[143] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imag. Sci.*, vol. 2, no. 1, pp. 183–202, 2009.

ABOUT THE AUTHORS

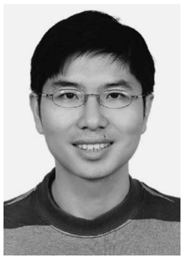
Guo-Xun Yuan received the B.S. degree in computer science from the National Tsinghua University, Hsinchu, Taiwan, and the M.S. degree in computer science from the National Taiwan University, Taipei, Taiwan. He is currently working towards the Ph.D. degree at the University of California Davis, Davis.

His research interest is large-scale data classification.



Chia-Hua Ho received the B.S. degree in computer science from the National Taiwan University, Taipei, Taiwan, in 2010, where he is currently working towards the M.S. degree at the Department of Computer Science.

His research interests are machine learning and data mining.



Chih-Jen Lin (Fellow, IEEE) received the B.S. degree in mathematics from the National Taiwan University, Taipei, Taiwan, in 1993 and the Ph.D. degree in industrial and operations engineering from the University of Michigan, Ann Arbor, in 1998.

He is currently a Distinguished Professor at the Department of Computer Science, National Taiwan University. His major research areas include machine learning, data mining, and numerical optimization. He is best known for his work on support vector machines (SVMs) for data classification. His software LIBSVM is one of the most widely used and cited SVM packages. Nearly all major companies apply his software for classification and regression applications.

Prof. Lin received many awards for his research work. A recent one is the ACM KDD 2010 best paper award. He is an Association for Computing Machinery (ACM) distinguished scientist for his contribution to machine learning algorithms and software design. More information about him and his software tools can be found at http://www.csie.ntu.edu.tw/~cjlin.