Recent Advances of Large-scale Linear Classification
Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin

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.

Index Terms—Large linear classification, Support vector machines, Logistic regression, Multi-class classification.

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 time is much shorter. Linear classifiers cover popular methods such as support vector machines (SVM) [1], [2], logistic regression (LR) [3] 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 multi-class problems, they mainly focused on nonlinear classification. In Section V we systematically compare methods for multi-class linear classification.

Linear classification can be further applied to many other scenarios. We investigate some examples in Section VII. In particular, we show that linear classifiers can be effectively employed to either directly or indirectly approximate nonlinear classifiers. In Section VII we 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 either try to reduce data reading time or take an online setting. 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 \((y_i, x_i) \in \{-1, +1\} \times \mathbb{R}^n, i = 1, \ldots, l\), where \(y_i\) is the label and \(x_i\) is the feature vector, some classification methods construct the following decision function.

\[ d(x) \equiv w^T \phi(x) + b, \quad (1) \]

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

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

\[ w \equiv \sum_{i=1}^{l} \alpha_i \phi(x_i) \text{ for some } \alpha \in \mathbb{R}^l, \quad (2) \]

and the following kernel function can be easily calculated

\[ K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j), \]

then the decision function can be calculated by

\[ d(x) \equiv \sum_{i=1}^{l} \alpha_i K(x_i, x) + b, \quad (3) \]

regardless of the dimensionality of \(\phi(x)\). For example,

\[ K(x_i, x_j) \equiv (x_i^T x_j + 1)^2 \quad (4) \]
is the degree-2 polynomial kernel with
\[ \phi(x) = [1, \sqrt{2}x_1, \ldots, \sqrt{2}x_n, x_1^2, \ldots, x_n^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_1x_n, \ldots, \sqrt{2}x_{n-1}x_n] \in \mathbb{R}^{(n+2)(n+1)/2}. \]
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 \( \phi \), the cost of predicting a testing instance \( x \) via \( \text{(3)} \) can be up to \( O(n) \). In contrast, without using kernels, \( w \) is available in an explicit form, so we can predict an instance by \( \text{(1)} \). With \( \phi(x) = x \),
\[ w^T \phi(x) = w^T x \]
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 RBF kernel). Table I shows the accuracy and training/testing time. Generally, nonlinear SVM has better accuracy, especially for problems MNIST, icsn11, covtype, and webspam. However, for problems with large numbers of features, i.e. news20, real-sim, and yahoo-japan, the accuracy values of linear and nonlinear SVMs are similar. Regarding training and testing time, Table I clearly indicates that linear classifiers are at least an order of magnitude faster.

For problems where linear classifiers yield comparable accuracy to nonlinear, they 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 \[ \text{[10]} \]. Each feature, corresponding to a word, 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 \( \text{(1)} \), linear classification involves the following risk minimization problem.
\[ \min_{w, b} \ f(w, b) \equiv r(w) + C \sum_{i=1}^{l} \xi(w, b; x_i, y_i), \]
where \( r(w) \) is the regularization term and \( \xi(w, b; x, y) \) is the loss function associated with the observation \((y, x)\). Parameter \( C > 0 \) is user-specified for balancing \( r(w) \) 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 \[ \text{[11]} \], \[ \text{[12]} \] showed similar performances with/without the bias term. Another setting is to put \( b \) into the regularization term by \( \|w^T b\| \) and add one constant feature by \( x_i^T \leftarrow [x_i^T, 1], \forall i \). In the rest of this paper, we omit the bias term \( b \), so \( \text{(6)} \) is simplified to
\[ \min_{w} \ f(w) \equiv r(w) + C \sum_{i=1}^{l} \xi(w; x_i, y_i) \]
and the decision function becomes \( d(x) \equiv w^T x \).

A. Support Vector Machines and Logistic Regression

In \[ \text{[7]} \], the loss function is used to penalize a wrongly-classified observation \((x, y)\). There are three common loss functions considered in the literature of linear classification.
\[ \xi_{L1}(w; x, y) \equiv \max(0, 1 - yw^T x), \]
\[ \xi_{L2}(w; x, y) \equiv \max(0, 1 - yw^T x)^2, \]
\[ \xi_{LR}(w; x, y) \equiv \log(1 + e^{-yw^T x}). \]
Eqs. \( \text{(8)} \) and \( \text{(9)} \) are referred to as L1 and L2 losses, respectively. Problem \( \text{(7)} \) using \( \text{(8)} \) and \( \text{(9)} \) as the loss function is often called L1-loss and L2-loss SVM, while problem \( \text{(7)} \) using \( \text{(10)} \) is referred to as logistic regression (LR). Both SVM and LR are popular classification methods. The three loss functions in \( \text{(8)}-\text{(10)} \) are all convex and non-negative. L1 loss is not differentiable at the point \( yw^T x = 1 \), while L2 loss is differentiable, but not twice differentiable \[ \text{[13]} \]. For logistic loss, it is twice differentiable. Figure 1 shows that these three losses are increasing functions of \(-yw^T x\). 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 \( x \). Overfitting training data to minimize 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.
\[ r_{L2}(w) \equiv \frac{1}{2} \|w\|_2^2 = \frac{1}{2} \sum_{j=1}^{n} w_j^2 \quad \text{and} \quad (11) \]
\[ r_{L1}(w) \equiv \|w\|_1 = \sum_{j=1}^{n} |w_j|. \quad \text{(12)} \]
Problem \( \text{(7)} \) with L2 regularization and L1 loss is the standard SVM proposed in \[ \text{[1]} \]. Both \( \text{(11)} \) and \( \text{(12)} \) are convex and separable functions. The effect of regularization on a variable is to push it toward zero. Then, the search space of \( w \) is more confined and overfitting may be avoided. It is known that
TABLE I

<table>
<thead>
<tr>
<th>Data set</th>
<th>#instances</th>
<th>#features</th>
<th>Linear Training</th>
<th>Nonlinear (kernel) Training</th>
<th>Nonlinear (kernel) Testing</th>
<th>Accuracy to nonlinear difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST38</td>
<td>11,982</td>
<td>1,984</td>
<td>0.1 0.00</td>
<td>96.82</td>
<td>38.1 5.61</td>
<td>99.70 -2.88</td>
</tr>
<tr>
<td>ijonn1</td>
<td>49,990</td>
<td>91,701</td>
<td>1.6 0.02</td>
<td>91.81</td>
<td>26.8 20.29</td>
<td>98.69 -6.88</td>
</tr>
<tr>
<td>covtype</td>
<td>464,810</td>
<td>116,202</td>
<td>1.4 0.02</td>
<td>76.37</td>
<td>46,695.8 1,311.20</td>
<td>96.11 -19.74</td>
</tr>
<tr>
<td>news20</td>
<td>15,997</td>
<td>3,999</td>
<td>1.1 0.02</td>
<td>96.95</td>
<td>383.2 100.38</td>
<td>96.90 0.05</td>
</tr>
<tr>
<td>real-sim</td>
<td>57,848</td>
<td>14,461</td>
<td>0.3 0.01</td>
<td>97.44</td>
<td>93.8 81.94</td>
<td>97.82 -0.38</td>
</tr>
<tr>
<td>yahoo-japan</td>
<td>140,963</td>
<td>35,240</td>
<td>3.1 0.04</td>
<td>92.63</td>
<td>20,955.2 1,890.83</td>
<td>93.31 -0.68</td>
</tr>
<tr>
<td>webspam</td>
<td>280,000</td>
<td>70,000</td>
<td>25.7 0.04</td>
<td>93.35</td>
<td>15,681.8 853.34</td>
<td>99.26 5.91</td>
</tr>
</tbody>
</table>

IV. TRAINING TECHNIQUES

To obtain the model \( w \), in the training phase we need to solve the convex optimization problem (7). While 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** In large-scale linear classification, data properties should be taken into account in the algorithm design. Algorithms which are efficient for some applications may not perform well in other areas. For example, some applications have much more instances than features, while some have the opposite. Other properties include the number of non-zero feature values, feature distribution, and 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 non-differentiable L1-regularized problems.

In some situations, by reformulation, we are able to transform a non-differentiable problem to be differentiable. For example, by letting \( w = w^+ + w^- \) (\( w^+, w^- \geq 0 \)), L1-regularized classifiers can be written as

\[
\begin{align*}
\min_{w^+, w^-} & \quad \sum_{i=1}^{n} \left( w^+_i \right) + \sum_{j=1}^{n} \left( w^-_j \right) + \frac{1}{2} \sum_{i=1}^{l} \left( w^+_i - w^-_i \right)^2 \quad \text{subject to} \quad w^+_i, w^-_i \geq 0, \quad j = 1, \ldots, n. \quad (14)
\end{align*}
\]

However, there is no guarantee that solving a differentiable form is better. Recent comparisons [12] show that for L1-regularized classifiers, methods that directly minimize the non-differentiable form are often faster than those solving (14).

• **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:

\[
\begin{align*}
\min_{\alpha} & \quad f^D(\alpha) \equiv \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\
\text{subject to} & \quad 0 \leq \alpha_i \leq C, \quad \forall i = 1, \ldots, l,
\end{align*}
\]

where \( Q_{ij} \equiv y_i y_j x_i^T x_j \). The objective function in (15) becomes smooth although the primal objective function is non-differentiable due to the L1 loss. Note that the primal optimal \( w \) and the dual optimal \( \alpha \) satisfy the relationship (2).

²Because the bias term \( b \) is not considered, therefore, different from the dual problem considered in SVM literature, an inequality constraint \( \sum_{i=1}^{l} y_i \alpha_i = 0 \) is absent from (15).

³However, we do not necessarily need the dual problem to get (2). For example, the reduced SVM [16] directly assumes that \( w \) is the linear combination of a subset of data.
However, the dual form may not be always easier to solve. For example, the dual form of L1-regularized problems involves general linear constraints, which are more complicated than bound constraints in \([15]\).

Dual problems come with another nice property that each variable \(\alpha_i\) corresponds to a training instance \((y_i, x_i)\). 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.

- **Using first-order or higher-order information** First-order methods, such as gradient descent 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.

Higher-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 higher-order methods are useful for applications needing to accurately solve the optimization problem \([7]\). Some (e.g., \([17]\)) have tried a hybrid setting by using first-order methods in the beginning and switching to higher-order methods in the end.

- **Cost 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, \([17], [18], [19]\).

- **Parallelization** Most existing training algorithms are inherently sequential, but a parallel algorithm can make good use of the computational power in a multi-core machine or a distributed system. However, the communication cost between different cores or nodes may become a new bottleneck.

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 \([20]\). However, these algorithms may not be suitable for sparse data with both large numbers of instances and features, for which we show in Section III 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\(^{perf}\) \([4]\), OCAS \([21]\), and BMRM \([22]\), a stochastic (sub-)gradient descent method for the primal form (Pegasos \([5]\) and SGD \([23]\), 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 \([18]\), a Newton method \([24]\), and a trust region Newton method (LIBLINEAR \([7]\)). For the dual problem, a coordinate descent method is in the software LIBLINEAR \([6]\).

- **L2-regularized LR:** Most works focus on the primal problem. They include iterative scaling methods \([11], [25], [26]\), a truncated Newton method \([27]\), and a trust region Newton method (LIBLINEAR \([28]\). Few works solve the dual problem. One example is a coordinate descent method (LIBLINEAR \([29]\)).

- **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 \([30]–[32]\).

- **L1-regularized L2-loss SVM:** Some proposed methods include a coordinate descent method (LIBLINEAR \([12]\)) and a Newton-type method \([19]\).

- **L1-regularized LR:** Most methods solve the primal form, e.g., an interior-point method (\(\mathsf{LIBLINEAR}\) \([23]\)), (block) coordinate descent methods (BBR \([34]\) and CGD \([35]\), a quasi-Newton method (OWL-QN \([36]\), Newton-type methods (GLMNET \([37]\) and LIBLINEAR \([19]\), and a Nesterov’s method (SLEP \([38]\). Recently, an augmented Lagrangian method (DAL \([19]\) is proposed for solving the dual problem. Comparisons of methods for L1-regularized LR include \([12], [40]\).

In the rest of this section, we show details of some representative optimization algorithms for large-scale linear classification.

### B. Example: A Sub-gradient Method (Pegasos)

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 VILA.

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

\[
\min_w \quad f(w; B) \equiv \frac{1}{2} \|w\|^2 + C \sum_{i \in B} \max(0, 1 - y_i \langle w, x_i \rangle).
\]

Here, for the deterministic setting, \(B\) is the whole training set. Because L1 loss is not differentiable, Pegasos takes the following sub-gradient direction of \(f(w; B)\):

\[
\nabla S f(w; B) \equiv w - C \sum_{i \in B^+} y_i x_i,
\]

where \(B^+ \equiv \{i \mid i \in B, 1 - y_i \langle w, x_i \rangle > 0\}\), and updates \(w\) by

\[
w \leftarrow w - \eta \nabla S f(w; B),
\]

where \(\eta = (C) / k\) is the learning rate and \(k\) is the iteration index. Different from earlier sub-gradient descent methods,
Then, by checking the ratio of actual function reduction to estimated function reduction, TRON decides if \( w \) should be updated and then adjusts \( \Delta \). A large enough \( \sigma \) indicates that the quadratic model \( q(d) \) is close to \( f(w + d) - f(w) \), so TRON updates \( w \) to be \( w + d \) and slightly enlarges the trust region interval \( \Delta \) for the next iteration. Otherwise, the current iterate \( w \) is unchanged and the trust region interval \( \Delta \) 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]**

- Given \( w, \Delta, \) and \( \sigma_0 \).
- For \( k = 1, 2, 3, \ldots \)
  a. Find an approximate solution \( d \) of (20) by the conjugate gradient method.
  b. Check the ratio \( \sigma \) in (21).
  c. If \( \sigma > \sigma_0 \)
    \[ w \leftarrow w + d. \]
  d. Adjust \( \Delta \) according to \( \sigma \).

If the loss function is not twice differentiable (e.g., L2 loss), we can use generalized Hessian [13] as \( \nabla^2 f(w) \) in (19).

Some difficulties of applying Newton methods to linear classification include that \( \nabla^2 f(w) \) may be a huge \( n \times n \) matrix and solving (20) is expensive. Fortunately, \( \nabla^2 f(w) \) of linear classification problems takes the following special form

\[
\nabla^2 f(w) = I + CX^T D_w X,
\]

where \( I \) is an identity matrix, \( X \equiv [x_1, \ldots, x_l]^T \), and \( D_w \) is a diagonal matrix. In [28], a conjugate gradient method is applied to solve (20), where the main operation is the product between \( \nabla^2 f(w) \) and a vector \( v \). By

\[
\nabla^2 f(w) v = v + C(X^T (D_w X)v),
\]

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

Because of using higher-order information (Newton directions), TRON gives fast local convergence. It has been extended to solve L1-regularized LR and L2-loss SVM in [12] by reformulating (7) to a bound-constrained optimization problem in (14).

**D. Example: Solving Dual SVM by Coordinate Descent Methods**

Hsieh et al. [6] proposed a coordinate descent method for the dual L2-regularized linear SVM in (15). Here, we focus on L1-loss SVM, although the 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 \( i \)-th variable, the following one-variable problem is solved.

\[
\min_d f^D(\alpha + de_i) - f^D(\alpha)
\]

subject to \( 0 \leq \alpha_i + d \leq C, \)

where \( e_i = [0, \ldots, 0, 1, 0, \ldots, 0]^T \) and \( i - 1 \)

\[
f^D(\alpha + de_i) - f^D(\alpha) = \frac{1}{2} Q_{ii} d^2 + \nabla_i f^D(\alpha)d.
\]

This simple quadratic function can be easily minimized. After considering the constraint, a simple update rule for \( \alpha_i \) is

\[
\alpha_i \leftarrow \min(\max(\alpha_i - \frac{\nabla_i f^D(\alpha)}{Q_{ii}}, 0), C).
\]

---

The optimal solution of \( f(w) \) is proven to be in the ball set \( \{ w \mid \|w\|_2 \leq \sqrt{C} \} \); see Theorem 1 in [5].
From (23), $Q_{ii}$ and $\nabla_i f^D(\alpha)$ are our needs. The diagonal entries of $Q_{ii}$, $Q_{ii}, \forall i$, are computed only once initially, but

$$\nabla_i f^D(\alpha) = (Q\alpha)_i - 1 = \sum_{t=1}^{l} (y_i y_t x_t^T x_i) \alpha_t - 1 \quad (24)$$

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

$$u \equiv \sum_{i=1}^{l} y_i \alpha_i x_i, \quad (25)$$

so that (24) becomes

$$\nabla_i f^D(\alpha) = (Q\alpha)_i - 1 = y_i u^T x_i - 1. \quad (26)$$

If $u$ is available through the training process, then the cost $O(nl)$ in (24) is significantly reduced to $O(n)$. The remaining task is to maintain $u$. Following (25), if $\alpha_i$ and $\alpha_t$ are values before and after the update (23), respectively, then we can easily maintain $u$ by the following $O(n)$ operation.

$$u \leftarrow u + y_i (\alpha_i - \alpha_t) x_i. \quad (27)$$

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

The vector $u$ defined in (25) is in the same form as $w$ in (2). In fact, as $\alpha$ approaches a dual optimal solution, $u$ will converge to the primal optimal $w$ following the primal-dual relationship.

The linear convergence of Algorithm 3 is established in [6] using techniques in [42]. They 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 [43], 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], [43]). 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 $u$ in (25) because $x_i, \forall i$ are available. For nonlinear SVM, $\nabla_i f^D(\alpha)$ in (24) 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. [37] is a Newton method for L1-regularized minimization. An improved version newGLMNET [19] is proposed for large-scale training.

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

- Given $\alpha$ and the corresponding $u = \sum_{i=1}^{l} y_i \alpha_i x_i$.
- Compute $Q_{ii}, \forall i = 1, \ldots, l$.
- While $\alpha$ is not optimal
  - For $i = 1, \ldots, l$
    - (a) Compute $G = y_i u^T x_i - 1$ in (20).
    - (b) $\alpha_i \leftarrow \alpha_i$.
    - (c) $\alpha_i \leftarrow \min(\max(\alpha_i - G/Q_{ii}, 0), C)$.
    - (d) $u \leftarrow u + y_i (\alpha_i - \alpha_t) x_i$.

newGLMNET minimizes $f(w)$ by a Newton method. Because the 1-norm term is not differentiable, we represent $f(w)$ as the sum of two terms $\|w\|_1 + L(w)$, where

$$L(w) \equiv C \sum_{i=1}^{l} \xi_i (w; x_i, y_i).$$

At each iteration, newGLMNET considers the second-order approximation of $L(w)$ and solves the following problem.

$$\min_d q(d) \equiv \|w + d\|_1 - \|w\|_1 + \nabla L(w)^T d + \frac{1}{2} d^T H d, \quad (28)$$

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

Although (28) is similar to (20), its optimization is more difficult because of the 1-norm term. Thus, newGLMNET further breaks (28) to sub-problems by a coordinate descent procedure. In a setting similar to the method in Section [IV-D] each time a one-variable function is minimized.

$$q(d + z e_j) - q(d) \equiv |w_j + d_j + z| - |w_j + d_j| + G_j z + \frac{1}{2} H_{jj} z^2, \quad (29)$$

where $G \equiv \nabla L(w)$ and $H$. This one-variable function (29) has a simple closed-form minimizer (see [44], [45], and Appendix B of [12]).

$$z = \begin{cases} \frac{-G_j + 1}{H_{jj}} & \text{if } G_j + 1 \leq H_{jj} (w_j + d_j), \\ \frac{-G_j - 1}{H_{jj}} & \text{if } G_j - 1 \geq H_{jj} (w_j + d_j), \\ - (w_j + d_j) & \text{otherwise.} \end{cases} \quad \text{(30)}$$

At each iteration of newGLMNET, the coordinate descent method does not solve problem (28) exactly. Instead, it designs an adaptive stopping condition so that initially problem (28) is solved loosely and in the final iterations, (28) is more accurately solved.

After an approximate solution $d$ of (28) is obtained, we need a line search procedure to ensure the sufficient function decrease. It finds $\lambda \in (0, 1]$ such that

$$f(w + \lambda d) - f(w) \leq \sigma \lambda \left( \|w + d\|_1 - \|w\|_1 + \nabla L(w)^T d \right), \quad \text{where } \sigma \in (0, 1).$$

The overall procedure of newGLMNET is in Algorithm 3.

Due to the adaptive setting, in the beginning, newGLMNET behaves like a coordinate descent method, which is able
to quickly obtain an approximate \( w \); however, in the final stage, the iterate \( w \) converges fast because a Newton step is taken. Recall in Section IV-A we mentioned that feature-wise data access is required for some primal approaches. Here, newGLMNET is an example. See details in [19] by checking how \( G_j \) in (29) is updated.

V. Multi-class Linear Classification

Most classification methods are originally proposed to solve a two-class problem; however, extensions of these methods to multi-class classification have been studied. For non-linear SVM, some works (e.g., [46], [47]) have comprehensively compared different multi-class solutions. In contrast, few studies have focused on multi-class linear classification. This section introduces and compares some commonly used methods.

A. Solving Several Binary Problems

Multi-class 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, [48], [49].

- One-against-rest method If there are \( k \) classes in the training data, the one-against-rest method [50] constructs \( k \) binary classification models. To obtain the \( m \) th model, instances from the \( m \) th class of the training set are treated as positive, and all other instances are negative. Then the weight vector \( w_m \) for the \( m \) th model can be generated by any linear classifier. After obtaining all \( k \) models, we say an instance \( x \) is in the \( m \) th class if the decision value (1) of the \( m \) th model is the largest, i.e.,

\[
\text{class of } x \equiv \arg \max_{m=1,...,k} w_m^T x. \tag{31}
\]

The cost is \( O(nk) \) for testing an instance.

- One-against-one method One-against-one method [51] 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 are \( k(k-1)/2 \) combination of two classes, \( k(k-1)/2 \) weight vectors are constructed: \( w_{1,2}, w_{1,3}, \ldots, w_{1,k}, w_{2,3}, \ldots, w_{(k-1),k}. \) There are different methods for testing. One approach is by voting [52]. For a testing instance \( x \), if model \((i,j)\) predicts \( x \) as in the \( i \)th class, then a counter for the \( i \)th class is added by one; otherwise, the counter for the \( j \)th class is added. Then we say \( x \) is in the \( i \)th class if the \( i \)th 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 [48], [49].

For linear classifiers, one-against-one method is shown to give better testing accuracy than one-against-rest [53]. 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 [46].

DAGSVM [54] is the same as one-against-one 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 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 multi-class classification [55], [56]. Here we discuss details of Crammer and Singer’s approach [56]. Assume class labels are \( 1, \ldots, k \). They consider the following optimization problem:

\[
\min_{w_1, \ldots, w_k} \frac{1}{2} \sum_{m=1}^{k} \| w_m \|^2 + C \sum_{i=1}^{l} \xi_C(\{w_m\}_{m=1}^{k}; x_i, y_i), \tag{32}
\]

where

\[
\xi_C(\{w_m\}_{m=1}^{k}; x_i, y_i) \equiv \max_{m \neq y_i} \max(0, 1 - (w_{y_i} - w_m)^T x_i). \tag{33}
\]

The setting is like to combine all binary models of the one-against-rest method. There are \( k \) weight vectors \( w_1, \ldots, w_k \) for \( k \) classes. In the loss function (33), for each \( m \), \( \max(0, 1 - (w_{y_i} - w_m)^T x_i) \) is similar to the L1 loss in (8) for binary classification. Overall, we hope that the decision value of \( x_i \) by the model \( w_{y_i} \) is at least one larger than the values by other models. For testing, the decision function is also (31).

Early works of this method focus on the nonlinear (i.e., kernel) case [46], [56], [57]. A study for linear classification is in [58], which applies a coordinate descent method to solve the dual problem of (32). The idea is similar to the method in Section IV-D, however, at each step, a larger sub-problem of \( k \) variables is solved. A nice property of this \( k \)-variable sub-problem is that it has a closed-form solution. Experiments in [58] show that solving (32) gives slightly better accuracy than one-against-rest, 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 has
made the approach of solving one single optimization problem less practical [46]. A careful implementation of the approach in [58] is given in [7, Appendix E].

C. Maximum Entropy

Maximum Entropy (ME) [59] is a generalization of logistic regression for multi-class problems [6] and a special case of conditional random fields [60] (see Section VII-A). It is widely applied by NLP applications. We still assume class labels $1, \ldots, k$ for an easy comparison to (32) in our subsequent discussion. ME models the following conditional probability function of label $y$ given data $x$.

$$P(y|x) \equiv \frac{\exp(w^T y x)}{\sum_m \exp(w^T m x)},$$

where $w_m, \forall m$ are weight vectors like those in (31) and (32). This model is also called multinomial logistic regression.

ME minimizes the following regularized negative log-likelihood.

$$\min_{w_1, \ldots, w_m} \frac{1}{2} \sum_{m=1}^{k} \|w_k\|^2 + C \sum_{i=1}^{l} \xi_{ME}(\{w_m\}_{m=1}^{k}; x_i, y_i),$$

(35)

where

$$\xi_{ME}(\{w_m\}_{m=1}^{k}; x, y) \equiv -\log P(y|x).$$

Clearly, (35) is similar to (32) and $\xi_{ME}(\cdot)$ can be considered as a loss function. If $w^T y x_i \gg w^T m x_i, \forall m \neq y_i$, then $\xi_{ME}(\{w_m\}_{m=1}^{k}; x_i, y_i)$ is close to zero (i.e., no loss). On the other hand, if $w^T y x_i$ is smaller than other $w^T m x_i, m \neq y_i$, then $P(y_i|x_i) \ll 1$ and the loss is large. For prediction, the decision function is also (31).

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

$$P(y|x) \equiv \frac{\exp(w^T f(x, y))}{\sum_y \exp(w^T f(x, y))}.$$  

(36)

Eq. (34) is a special case of (36) by

$$f(x_i, y) = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} y - 1 \in \mathbb{R}^{nk} \text{ and } w = \begin{bmatrix} w_1 \\ \vdots \\ w_k \end{bmatrix}.$$  

(37)

Many studies have investigated optimization methods for L2-regularized ME. For example, Malouf [61] compares iterative scaling methods [62], gradient descent, conjugate gradient descent, and quasi-Newton method to solve (35). Experiments show that quasi Newton performs better. In [11], a framework is proposed to explain variants of iterative scaling methods [25], [62], [63] and make a connection to coordinate descent methods. For L1-regularized ME, [36] proposes an extension of L-BFGS.

Recently, instead of solving the primal problem (35), some works solve the dual problem. A detailed derivation of the dual ME is in [29, Appendix A.7]. Memisevic [64] proposed a two-level decomposition method. Similar to the coordinate descent method [58] for (32) in Section V-B, in [64], a sub-problem of $k$ variables is considered at a time. However, the sub-problem does not have a closed-form solution, so a second-level coordinate descent method is applied. Collin et al. [65] proposed an exponential gradient method to solve ME dual. They also decompose the problem into $k$-variable sub-problems, but only approximately solve each sub-problem. The work in [29] follows [64] 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 II. Clearly, one-against-one and DAgSVM are less practical because of the much higher storage, although the comparison in [53] indicates that one-against-one gives slightly better testing accuracy. Note that the situation is very different for the kernel case [46], where one-against-one and DAgSVM are very useful methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Storage</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-against-rest</td>
<td>$O(kn)$</td>
<td>$O(kn)$</td>
</tr>
<tr>
<td>one-against-one</td>
<td>$O(k^2n)$</td>
<td>$O(k^2n)$</td>
</tr>
<tr>
<td>DAgSVM</td>
<td>$O(k^2n)$</td>
<td>$O(kn)$</td>
</tr>
<tr>
<td>Crammer and Singer’s</td>
<td>$O(kn)$</td>
<td>$O(kn)$</td>
</tr>
<tr>
<td>maximum entropy</td>
<td>$O(kn)$</td>
<td>$O(kn)$</td>
</tr>
</tbody>
</table>

TABLE II

COMPARISON OF METHODS FOR MULTI-CLASS LINEAR CLASSIFICATION

IN STORAGE (MODEL SIZE) AND TESTING TIME. $n$ IS THE NUMBER OF FEATURES AND $k$ IS THE NUMBER OF CLASSES.
is used for training linear/kernelized classifiers and $\phi(x) \in \mathbb{R}^d$. From Section VII-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], they 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 $O(n^2)$ versus $O(nl)$, where $n$ is the average number of non-zero values per instance. For large sparse data sets, $n \ll l$, so their approach can be very efficient. For dependency parsing problems in NLP, they show 50 times faster training than using kernel methods.

A general framework was proposed in [69] for various nonlinear mappings of data. They noticed that to perform the coordinate descent method in Section VII-D one only needs that $u^T \phi(x)$ in (26) and $u \leftarrow u + y_i(\alpha_i - \bar{\alpha}_i)\phi(x)$ in (27) can be performed. Thus, even if $\phi(x)$ cannot be explicitly represented, as long as these two operations can be performed, Algorithm VII-D is applicable.

Studies in [70], [71] 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 $\phi(x_i), \forall i$ explicitly, so they obtain the same model as a kernel method under $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$. However, they have limitations when the dimensionality of $\phi(x)$ 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 [72], we categorize these methods to the following two types.

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

  \[ \bar{Q} = \bar{\Phi} \bar{\Phi}^T \approx Q. \] (38)

  Assume $\bar{\Phi} \bar{\Phi}^T \equiv [\bar{x}_1, \ldots, \bar{x}_l]$. If we replace $Q$ in (15) with $\bar{Q}$, then (15) becomes the dual problem of training a linear SVM on the new set $(y_i, \bar{x}_i), i = 1, \ldots, l$. 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 (38) does not directly reveal how to adjust the decision function $\bar{\phi}(x)$.

  Early developments focused on finding a good approximation matrix $\Phi$. Some examples include Nyström method [73], [74] and incomplete Cholesky factorization [75], [76].

  Some works (e.g., [16]) consider approximations other than (38), but also lead to linear classification problems. A recent study [72] 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 Section 2.4 of [72]. 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 $\tilde{\phi} : \mathbb{R}^n \rightarrow \mathbb{R}^d$ such that

  \[ \tilde{\phi}(x)^T \tilde{\phi}(t) \approx K(x, t). \]

  Then, linear classifiers can be applied to new data $\tilde{\phi}(x_1), \ldots, \tilde{\phi}(x_l)$. The testing phase is straightforward because the mapping $\tilde{\phi}(\cdot)$ is available.

  Many mappings have been proposed. Examples include random Fourier projection [77], random projections [78], [79], polynomial approximation [80], and hashing [81–84]. 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., [77]) generate dense $\phi(x)$ vectors, while others give sparse vectors (e.g., [79]). A recent study focusing on the linear classification after obtaining $\tilde{\phi}(x_i), \forall i$ is in [85].

VII. TRAINING LARGE DATA BEYOND THE MEMORY 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 frequently accessing data from disk. Indeed, even if the memory is enough, loading data to memory may take more time than subsequent computation [86]. Therefore, the design of algorithms for data larger than memory is very different from that of traditional algorithms.

If the data set is even beyond the disk capacity of a single computer, then it must be stored distributedly. 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 [87], [88], 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. 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 \( w \) 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, they are also suitable for streaming data where each training instance is used only once. One of the popular online algorithm is the stochastic gradient descent method (SGD), which can be traced back to stochastic approximation method [89], [80]. Take the primal L2-regularized L1-loss SVM in [7] as an example. At each step, a training instance \( x_i \) is chosen and \( w \) is updated by

\[
  w \leftarrow w - \eta \nabla S \left( \frac{1}{2} \| w \|^2 + C \max(0, 1 - y_i w^T x_i) \right), \tag{39}
\]

where \( \nabla S \) is a sub-gradient operator and \( \eta \) is the learning rate. Specifically, (39) becomes the following update rule.

If \( 1 - y_i w^T x_i > 0 \), then

\[
  w \leftarrow (1 - \eta)w + \eta C y_i x_i. \tag{40}
\]

The learning rate \( \eta \) is gradually reduced along iterations.

It is well known that stochastic gradient descent 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, [91], [92]. For data with many features, recent studies [5], [23] 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 Theorem 2 in [5]).

Instead of solving the primal problem, we can design an online algorithm to solve the dual problem [9], [93]. For example, the coordinate descent method in Algorithm 2 can be easily extended to an online setting by replacing the sequential selection of variables with a random selection. Notice that the update rule (27) is similar to (40), but has the advantage of not needing to decide the learning rate \( \eta \). This online setting falls into the general framework of randomized coordinate descent methods in [94], [95]. Using the proof in [94], the linear convergence in expectation is obtained in Appendix 7.5 of [6].

To improve the convergence of SGD, some [96], [97] have proposed using higher-order information. The rule in (39) is replaced by

\[
  w \leftarrow w - \eta H \nabla S (\cdot), \tag{41}
\]

where \( H \) is an approximation of the inverse Hessian \( \nabla^2 f(w)^{-1} \). To save the cost at each update, practically \( H \) is a diagonal scaling matrix. Experiments [96], [97] show that using (41) is faster than (39).

The update rule in (39) 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 \( w \). To address this problem, recently several approaches have been proposed [98]–[103]. The stochastic coordinate descent method in [99] has been extended to a parallel version [104].

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 [105] 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 [86], which will be discussed in Section VII-C. Currently, Vowpal-Wabbit (version 5.1) supports unregularized linear classification and regression. It is extended to solve L1-regularized problems in [98].

We are aware that some 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 [106].

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., [86], [107]) have checked the situation that data are larger than memory but smaller than disk. Because of expensive disk 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 [86] 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 (15), sequentially one block of data in \( B \) is read and the following function of \( d \) is minimized under the condition \( 0 \leq \alpha_i + d_i \leq C, \forall i \in B \) and \( d_i = 0, \forall i \notin B \).

\[
  f^D (\alpha + d) - f^D (\alpha) = \frac{1}{2} d^T_B Q_B B d_B + d^T_B (Q \alpha - e) B
\]

\[
  = \frac{1}{2} d^T_B Q_B B d_B + \sum_{i \in B} y_i d_i (u^T_i x_i) - d^T_B e_B, \tag{42}
\]

where \( Q_B B \) is a sub-matrix of \( Q \) and \( u \) is defined in (25). By maintaining \( u \) in a way similar to (27), Eq. (42) involves only data in the block \( B \), which can be stored in memory. Eq. (42) can be minimized by any traditional algorithm. Experiments in [86] demonstrate that they can train data 20 times larger than the memory capacity. This method is extended in [108] 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], [43].
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 [109] selects a subset by reading data from disk only once. For data in a distributed environment, sub-sampling can be a complicated operation. Moreover, a subset fitting the memory of one single computer may be too small to give good accuracy.

Bagging [110] 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 distributedly stored data because we can directly consider data on each node as a subset. However, if data quality on 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 [111].

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 assume that the label \( y_i \) is a single value. For binary classification, it is +1 or −1, while for multi-class 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 \( (y_i, x_i) \in Y^{n_i} \times X^n, \forall i = 1, \ldots, l \), where \( x_i \) is the \( i \)th sentence, \( y_i \) 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 \( i \)th sentence. Note that we may not be able to split the problem to several independent ones by treating each value \( y_{ij} \) of \( y_i \) as the label, because \( y_{ij} \) not only depends on the sentence \( x_i \) but also other tags \( \langle y_{(j-1)}, \ldots, y_{(j+1)} \rangle \). To handle these problems, we could use structured learning models, e.g., conditional random fields [60] and structured SVM [112, 113].

• **Conditional Random Fields** Conditional random fields (CRF) [60] is a linear structured model commonly used in NLP. Using notation mentioned above and a feature function \( f(x, y) \) like ME, CRF solves the following problem:

\[
\min_w \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^l \xi_{\text{CRF}}(w; x_i, y_i),
\]

where

\[
\xi_{\text{CRF}}(w; x_i, y_i) = -\log P(y_i | x_i), \quad \text{and}
\]

\[
P(y_i | x) = \frac{\exp(w^T f(x, y_i))}{\sum_{y} \exp(w^T f(x, y))}. \tag{44}
\]

If elements in \( y_i \) are independent to each other, then CRF reduces to ME.

The optimization of (43) is challenging because in the probability model (44), the number of possible \( y \)'s is exponentially many. An important property to make CRF practical is that the gradient of the objective function in (43) can be efficiently evaluated by dynamic programming [60]. Some available optimization methods include L-BFGS (quasi Newton) and conjugate gradient [114], stochastic gradient descent [115], stochastic quasi Newton [26, 116], and trust region Newton method [117]. It is shown in [117] that the Hessian vector product (22) of the Newton method can also be evaluated by dynamic programming.

• **Structured SVM** Structured SVM solves the following optimization problem generalized form multi-class SVM in [35, 56].

\[
\min_w \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^l \xi_{\text{SS}}(w; x_i, y_i), \tag{45}
\]

where

\[
\xi_{\text{SS}}(w; x_i, y_i) \equiv \max_{y \neq y_i} \left( \max (0, \Delta(y_i, y) - w^T (f(x_i, y_i) - f(x_i, y))) \right),
\]

and \( \Delta(c) \) is a distance function with \( \Delta(y_i, y_i) = 0 \) and \( \Delta(y_i, y_j) = \Delta(y_j, y_i) \). Similar to the relation between conditional random fields and maximum entropy, if

\[
\Delta(y_i, y_j) = \begin{cases} 
0 & \text{if } y_i = y_j \\
1 & \text{otherwise}
\end{cases}
\]

and \( y_i \in \{1, \ldots, k\}, \forall i \), then structured SVM becomes Crammer and Singer’s problem in [32] following the definition of \( f(x, y) \) and \( w \) in (47)

Like CRF, the main difficulty to solve (45) is on handling an exponential number of \( y \) values. Some works (e.g., [22, 112, 118]) use a cutting plane method [119] to solve (45). In [120], stochastic subgradient descent method is applied on the primal problem for both online and batch setting.

B. Regression

Given training data \( \{(z_i, x_i)\}_{i=1}^n \subset \mathbb{R} \times \mathbb{R}^n \), a regression problem finds a weight vector \( w \) such that \( w^T x_i \approx z_i, \forall i \).

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

\[
\xi_{\text{LS}}(w; x, z) \equiv \frac{1}{2} (z - w^T x)^2 \quad \text{and}
\]

\[
\xi_r(w; x, z) \equiv \max(0, |z - w^T x| - \epsilon). \tag{47}
\]
The least square loss in $\ell_2$ is widely used in many places, while the $\epsilon$-insensitive loss in $\ell_\infty$ is extended from the L1 loss in $\ell_1$, where there is a user-specified parameter $\epsilon$ as the error tolerance. Problem (7) with L2 regularization and $\epsilon$-insensitive loss is called support vector regression (SVR) [21]. 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., [45], [122]–[129]) have been proposed. However, as pointed out in [12], 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 $x_i, \forall i$ 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. CONCLUSIONS

In this article, 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.

X. ACKNOWLEDGMENTS

This work was supported in part by the National Science Council of Taiwan via the grant 98-2221-E-002-136-MY3.

REFERENCES


