Journal of Machine Learning Research 9 (2008) 1871-1874 Submitted 5/08; Published 8/08

# LIBLINEAR: A Library for Large Linear Classification

**Rong-En Fan** B90098@CSIE.NTU.EDU.TW
**Kai-Wei Chang** B92084@CSIE.NTU.EDU.TW
**Cho-Jui Hsieh** B92085@CSIE.NTU.EDU.TW
**Xiang-Rui Wang** R95073@CSIE.NTU.EDU.TW
**Chih-Jen Lin** CJLIN@CSIE.NTU.EDU.TW
*Department of Computer Science*
*National Taiwan University*
*Taipei 106, Taiwan*
**Last modified: October 21, 2020**

**Editor:** Soeren Sonnenburg

## Abstract

LIBLINEAR is an open source library for large-scale linear classification. It supports logistic regression and linear support vector machines. We provide easy-to-use command-line tools and library calls for users and developers. Comprehensive documents are available for both beginners and advanced users. Experiments demonstrate that LIBLINEAR is very efficient on large sparse data sets.

**Keywords:** large-scale linear classification, logistic regression, support vector machines, open source, machine learning

## 1. Introduction

Solving large-scale classification problems is crucial in many applications such as text classification. Linear classification has become one of the most promising learning techniques for large sparse data with a huge number of instances and features. We develop LIBLINEAR as an easy-to-use tool to deal with such data. It supports L2-regularized logistic regression (LR), L2-loss and L1-loss linear support vector machines (SVMs) (Boser et al., 1992). It inherits many features of the popular SVM library LIBSVM (Chang and Lin, 2011) such as simple usage, rich documentation, and open source license (the BSD license[1]). LIBLINEAR is very efficient for training large-scale problems. For example, it takes only several *seconds* to train a text classification problem from the Reuters Corpus Volume 1 (rcv1) that has more than 600,000 examples. For the same task, a general SVM solver such as LIBSVM would take several hours. Moreover, LIBLINEAR is competitive with or even faster than state of the art linear classifiers such as Pegasos (Shalev-Shwartz et al., 2007) and SVM$^{\text{perf}}$ (Joachims, 2006). The software is available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear`.

This article is organized as follows. In Sections 2 and 3, we discuss the design and implementation of LIBLINEAR. We show the performance comparisons in Section 4. Closing remarks are in Section 5.

1. The New BSD license approved by the Open Source Initiative.

## 2. Large Linear Classification (Binary and Multi-class)

LIBLINEAR supports two popular binary linear classifiers: LR and linear SVM. Given a set of instance-label pairs $(\boldsymbol{x}_i, y_i), i = 1, \ldots, l,\ \boldsymbol{x}_i \in R^n,\ y_i \in \{-1, +1\}$, both methods solve the following unconstrained optimization problem with different loss functions $\xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i)$:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} \xi(\boldsymbol{w}; \boldsymbol{x}_i, y_i), \tag{1}$$

where $C > 0$ is a penalty parameter. For SVM, the two common loss functions are $\max(1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0)$ and $\max(1-y_i\boldsymbol{w}^T\boldsymbol{x}_i, 0)^2$. The former is referred to as L1-SVM, while the latter is L2-SVM. For LR, the loss function is $\log(1+e^{-y_i\boldsymbol{w}^T\boldsymbol{x}_i})$, which is derived from a probabilistic model. In some cases, the discriminant function of the classifier includes a bias term, $b$. LIBLINEAR handles this term by augmenting the vector $\boldsymbol{w}$ and each instance $\boldsymbol{x}_i$ with an additional dimension: $\boldsymbol{w}^T \leftarrow [\boldsymbol{w}^T, b], \boldsymbol{x}_i^T \leftarrow [\boldsymbol{x}_i^T, B]$, where $B$ is a constant specified by the user. See Appendix A for details.[2] The approach for L1-SVM and L2-SVM is a coordinate descent method (Hsieh et al., 2008). For LR and also L2-SVM, LIBLINEAR implements a Newton method (Galli and Lin, 2020). The Appendix of our SVM guide[3] discusses when to use which method. In the testing phase, we predict a data point $\boldsymbol{x}$ as positive if $\boldsymbol{w}^T\boldsymbol{x} > 0$, and negative otherwise. For multi-class problems, we implement the one-vs-the-rest strategy and a method by Crammer and Singer. Details are in Keerthi et al. (2008).

## 3. The Software Package

The LIBLINEAR package includes a library and command-line tools for the learning task. The design is highly inspired by the LIBSVM package. They share similar usage as well as application program interfaces (APIs), so users/developers can easily use both packages. However, their models after training are quite different (in particular, LIBLINEAR stores $\boldsymbol{w}$ in the model, but LIBSVM does not.). Because of such differences, we decide not to combine these two packages together. In this section, we show various aspects of LIBLINEAR.

### 3.1 Practical Usage

To illustrate the training and testing procedure, we take the data set news20,[4] which has more than one million features. We use the default classifier L2-SVM.

```
$ train news20.binary.tr
[output skipped]
$ predict news20.binary.t news20.binary.tr.model prediction
Accuracy = 96.575% (3863/4000)
```

The whole procedure (training and testing) takes less than 15 seconds on a modern computer. The training time without including disk I/O is less than one second. Beyond this

2. After version 2.40, for some optimization methods implemented in LIBLINEAR, users can choose whether the bias term should be regularized.
3. The guide can be found at `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.
4. This is the news20.binary set from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`. We use a 80/20 split for training and testing.

simple way of running LIBLINEAR, several parameters are available for advanced use. For example, one may specify a parameter to obtain probability outputs for logistic regression. Details can be found in the `README` file.

### 3.2 Documentation

The LIBLINEAR package comes with plenty of documentation. The `README` file describes the installation process, command-line usage, and the library calls. Users can read the "Quick Start" section, and begin within a few minutes. For developers who use LIBLINEAR in their software, the API document is in the "Library Usage" section. All the interface functions and related data structures are explained in detail. Programs `train.c` and `predict.c` are good examples of using LIBLINEAR APIs. If the `README` file does not give the information users want, they can check the online FAQ page.[5] In addition to software documentation, theoretical properties of the algorithms and comparisons to other methods are in Lin et al. (2008) and Hsieh et al. (2008). The authors are also willing to answer any further questions.

### 3.3 Design

The main design principle is to keep the whole package as simple as possible while making the source codes easy to read and maintain. Files in LIBLINEAR can be separated into source files, pre-built binaries, documentation, and language bindings. All source codes follow the C/C++ standard, and there is no dependency on external libraries. Therefore, LIBLINEAR can run on almost every platform. We provide a simple `Makefile` to compile the package from source codes. For Windows users, we include pre-built binaries.

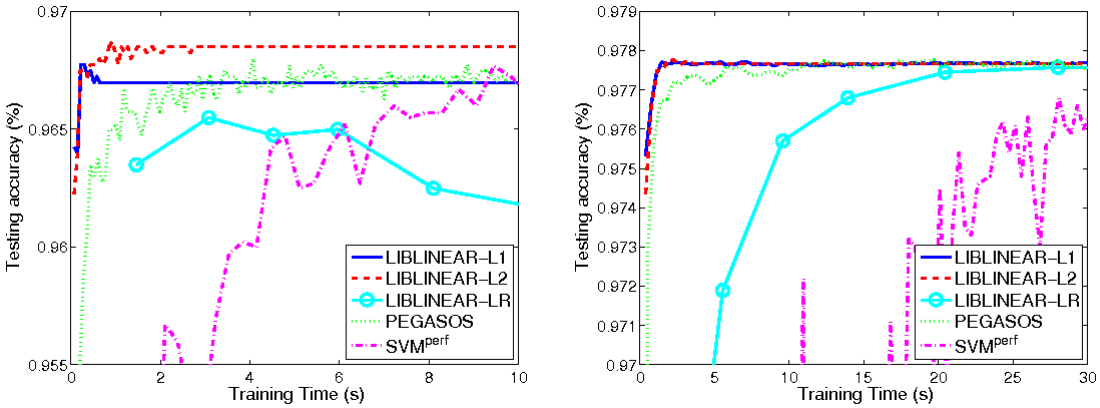
Library calls are implemented in the file `linear.cpp`. The `train()` function trains a classifier on the given data and the `predict()` function predicts a given instance. To handle multi-class problems via the one-vs-the-rest strategy, `train()` conducts several binary classifications, each of which is by calling the `train_one()` function. `train_one()` then invokes the solver of users' choice. Implementations follow the algorithm descriptions in Lin et al. (2008) and Hsieh et al. (2008). As LIBLINEAR is written in a modular way, a new solver can be easily plugged in. This makes LIBLINEAR not only a machine learning tool but also an experimental platform.

Making extensions of LIBLINEAR to languages other than C/C++ is easy. Following the same setting of the LIBSVM MATLAB/Octave interface, we have a MATLAB/Octave extension available within the package. Many tools designed for LIBSVM can be reused with small modifications. Some examples are the parameter selection tool and the data format checking tool.

## 4. Comparison

Due to space limitation, we skip here the full details, which are in Lin et al. (2008) and Hsieh et al. (2008). We only demonstrate that LIBLINEAR quickly reaches the testing accuracy corresponding to the optimal solution of (1). We conduct five-fold cross validation to select the best parameter $C$ for each learning method (L1-SVM, L2-SVM, LR); then we train on the whole training set and predict the testing set. Figure 1 shows the comparison between

5. FAQ can be found at `http://www.csie.ntu.edu.tw/~cjlin/liblinear/FAQ.html`.




(a) news20, $l$: 19,996, $n$: 1,355,191, #nz: 9,097,916 (b) rcv1, $l$: 677,399, $n$: 47,236, #nz: 156,436,656

Figure 1: Testing accuracy versus training time (in seconds). Data statistics are listed after the data set name. $l$: number of instances, $n$: number of features, #nz: number of nonzero feature values. We split each set to 4/5 training and 1/5 testing.

LIBLINEAR and two state of the art L1-SVM solvers: Pegasos (Shalev-Shwartz et al., 2007) and SVM$^{\text{perf}}$ (Joachims, 2006). Clearly, LIBLINEAR is efficient.

To make the comparison reproducible, codes used for experiments in Lin et al. (2008) and Hsieh et al. (2008) are available at the LIBLINEAR web page.

## 5. Conclusions

LIBLINEAR is a simple and easy-to-use open source package for large linear classification. Experiments and analysis in Lin et al. (2008), Hsieh et al. (2008) and Keerthi et al. (2008) conclude that solvers in LIBLINEAR perform well in practice and have good theoretical properties. LIBLINEAR is still being improved by new research results and suggestions from users. The ultimate goal is to make easy learning with huge data possible.

## References

B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, 1992.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM TIST*, 2(3):27:1–27:27, 2011.

Leonardo Galli and Chih-Jen Lin. Truncated Newton methods for linear classification. Technical report, National Taiwan University, 2020. URL `https://www.csie.ntu.edu.tw/~cjlin/papers/tncg/tncg.pdf`.

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 *ICML*, 2008.

T. Joachims. Training linear SVMs in linear time. In *KDD*, 2006.

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 *KDD*, 2008.

C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region Newton method for large-scale logistic regression. *JMLR*, 9:627–650, 2008.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: primal estimated sub-gradient solver for SVM. In *ICML*, 2007.

## Acknowledgments

This work was supported in part by the National Science Council of Taiwan via the grant NSC 95-2221-E-002-205-MY3.

## Appendix: Implementation Details and Practical Guide

## Appendix A. Formulations

This section briefly describes classifiers supported in LIBLINEAR. Given training vectors $\boldsymbol{x}_i \in R^n, i = 1, \ldots, l$ in two class, and a vector $\boldsymbol{y} \in R^l$ such that $y_i = \{1, -1\}$, a linear classifier generates a weight vector $\boldsymbol{w}$ as the model. The decision function is

$$\operatorname{sgn}\left(\boldsymbol{w}^T \boldsymbol{x}\right).$$

### A.1 Some Notes on the Bias Term

Before presenting all formulations, we briefly discuss the issue of the bias term in the model. Traditionally in classifiers such as SVM, the discriminant function of the classifier includes a bias term, $b$. For high-dimensional data it is known that with/without a bias term give similar performances. Therefore, **the default setting of LIBLINEAR does not consider a bias term in the model.**

For users stressing on having a bias term in the model, LIBLINEAR handles this term by augmenting the vector $\boldsymbol{w}$ and each instance $\boldsymbol{x}_i$ with an additional dimension: $\boldsymbol{w}^T \leftarrow [\boldsymbol{w}^T, b], \boldsymbol{x}_i^T \leftarrow [\boldsymbol{x}_i^T, B]$, where $B$ is a constant specified by the user. The optimization problem becomes

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \frac{1}{2}b^2 + C\sum\nolimits_{i=1}^{l} \xi(\begin{bmatrix}\boldsymbol{w}\\ b\end{bmatrix}; \begin{bmatrix}\boldsymbol{x}_i\\ B\end{bmatrix}, y_i). \tag{2}$$

After version 2.40, for some optimization methods, users can choose not to regularize the bias term. Then the optimization problem is

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum\nolimits_{i=1}^{l} \xi(\begin{bmatrix}\boldsymbol{w}\\ b\end{bmatrix}; \begin{bmatrix}\boldsymbol{x}_i\\ B\end{bmatrix}, y_i). \tag{3}$$

In the following subsections, we only consider the formulation in which the bias term is not included.

### A.2 L2-regularized L1- and L2-loss Support Vector Classification

L2-regularized L1-loss SVC solves the following primal problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}(\max(0, 1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i)),$$

whereas L2-regularized L2-loss SVC solves the following primal problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}(\max(0, 1 - y_i\boldsymbol{w}^T\boldsymbol{x}_i))^2. \tag{4}$$

Their dual forms are:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T\bar{Q}\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le U,\ i = 1, \ldots, l. \end{aligned}$$

where $\boldsymbol{e}$ is the vector of all ones, $\bar{Q} = Q + D$, $D$ is a diagonal matrix, and $Q_{ij} = y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$. For L1-loss SVC, $U = C$ and $D_{ii} = 0,\ \forall i$. For L2-loss SVC, $U = \infty$ and $D_{ii} = 1/(2C),\ \forall i$.

### A.3 L2-regularized Logistic Regression

L2-regularized LR solves the following unconstrained optimization problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l} \log(1 + e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}). \tag{5}$$

Its dual form is:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} + \sum_{i:\alpha_i > 0} \alpha_i \log \alpha_i + \sum_{i:\alpha_i < C} (C - \alpha_i)\log(C - \alpha_i) - \sum_{i=1}^{l} C \log C \\ \text{subject to} \quad & 0 \le \alpha_i \le C,\ i = 1, \ldots, l. \end{aligned} \tag{6}$$

### A.4 L1-regularized L2-loss Support Vector Classification

L1 regularization generates a sparse solution $\boldsymbol{w}$. L1-regularized L2-loss SVC solves the following primal problem:

$$\min_{\boldsymbol{w}} \quad \|\boldsymbol{w}\|_1 + C\sum_{i=1}^{l} (\max(0, 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i))^2. \tag{7}$$

where $\|\cdot\|_1$ denotes the 1-norm.

### A.5 L1-regularized Logistic Regression

L1-regularized LR solves the following unconstrained optimization problem:

$$\min_{\boldsymbol{w}} \quad \|\boldsymbol{w}\|_1 + C\sum_{i=1}^{l} \log(1 + e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}). \tag{8}$$

where $\|\cdot\|_1$ denotes the 1-norm.

### A.6 L2-regularized L1- and L2-loss Support Vector Regression

Support vector regression (SVR) considers a problem similar to (1), but $y_i$ is a real value instead of $+1$ or $-1$. L2-regularized SVR solves the following primal problems:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \begin{cases} C\sum_{i=1}^{l}(\max(0, |y_i - \boldsymbol{w}^T\boldsymbol{x}_i| - \epsilon)) & \text{if using L1 loss,} \\ C\sum_{i=1}^{l}(\max(0, |y_i - \boldsymbol{w}^T\boldsymbol{x}_i| - \epsilon))^2 & \text{if using L2 loss,} \end{cases}$$

where $\epsilon \geq 0$ is a parameter to specify the sensitiveness of the loss.

Their dual forms are:

$$\begin{aligned} \min_{\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-} \quad & \frac{1}{2} \begin{bmatrix} \boldsymbol{\alpha}^+ & \boldsymbol{\alpha}^- \end{bmatrix} \begin{bmatrix} \bar{Q} & -Q \\ -Q & \bar{Q} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}^+ \\ \boldsymbol{\alpha}^- \end{bmatrix} - \boldsymbol{y}^T(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-) + \epsilon \boldsymbol{e}^T(\boldsymbol{\alpha}^+ + \boldsymbol{\alpha}^-) \\ \text{subject to} \quad & 0 \leq \alpha_i^+, \alpha_i^- \leq U,\ i = 1, \ldots, l, \end{aligned} \tag{9}$$

where $\boldsymbol{e}$ is the vector of all ones, $\bar{Q} = Q + D$, $Q \in R^{l \times l}$ is a matrix with $Q_{ij} \equiv \boldsymbol{x}_i^T \boldsymbol{x}_j$, $D$ is a diagonal matrix,

$$D_{ii} = \begin{cases} 0 \\ \frac{1}{2C} \end{cases}, \text{ and } U = \begin{cases} C & \text{if using L1-loss SVR,} \\ \infty & \text{if using L2-loss SVR.} \end{cases}$$

Rather than (9), in LIBLINEAR, we consider the following problem.

$$\begin{aligned} \min_{\boldsymbol{\beta}} \quad & \frac{1}{2} \boldsymbol{\beta}^T \bar{Q} \boldsymbol{\beta} - \boldsymbol{y}^T \boldsymbol{\beta} + \epsilon \|\boldsymbol{\beta}\|_1 \\ \text{subject to} \quad & -U \leq \beta_i \leq U,\ i = 1, \ldots, l, \end{aligned} \tag{10}$$

where $\boldsymbol{\beta} \in R^l$ and $\|\cdot\|_1$ denotes the 1-norm. It can be shown that an optimal solution of (10) leads to the following optimal solution of (9).

$$\alpha_i^+ \equiv \max(\beta_i, 0) \quad \text{and} \quad \alpha_i^- \equiv \max(-\beta_i, 0).$$

## Appendix B. L2-regularized L1- and L2-loss SVM (Solving Dual)

See Hsieh et al. (2008) for details of a dual coordinate descent method.

## Appendix C. L2-regularized Logistic Regression (Solving Primal)

From versions 1.0 to 2.30, a trust region Newton method was considered. Some details are as follows.

- The main algorithm was developed in Lin et al. (2008).
- After version 2.11, the trust-region update rule is improved by the setting proposed in Hsia et al. (2017).
- After version 2.20, the convergence of conjugate gradient method is improved by applying a preconditioning technique proposed in Hsia et al. (2018).

After version 2.40, the trust-region Newton method is replaced by a line-search Newton method. Details are in Galli and Lin (2020a) and the release notes of LIBLINEAR 2.40 (Galli and Lin, 2020b).

## Appendix D. L2-regularized L2-loss SVM (Solving Primal)

The algorithm is the same as the Newton methods for logistic regression (trust-region Newton in Lin et al. (2008) before version 2.30 and line-search Newton in Galli and Lin (2020a) after version 2.40). The only difference is the formulas of gradient and Hessian-vector products. We list them here.

The objective function is in (4). Its gradient is

$$\boldsymbol{w} + 2CX_{I,:}^T(X_{I,:}\boldsymbol{w} - \boldsymbol{y}_I), \tag{11}$$

where $I \equiv \{i \mid 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i > 0\}$ is an index set, $\boldsymbol{y} = [y_1, \ldots, y_l]^T$, and $X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_l^T \end{bmatrix}$.

Eq. (4) is differentiable but not twice differentiable. To apply the Newton method, we consider the following *generalized Hessian* of (4):

$$\mathcal{I} + 2CX^T DX = \mathcal{I} + 2CX_{I,:}^T D_{I,I} X_{I,:}, \tag{12}$$

where $\mathcal{I}$ is the identity matrix and $D$ is a diagonal matrix with the following diagonal elements:

$$D_{ii} = \begin{cases} 1 & \text{if } i \in I, \\ 0 & \text{if } i \notin I. \end{cases}$$

The Hessian-vector product between the generalized Hessian and a vector $\boldsymbol{s}$ is:

$$\boldsymbol{s} + 2CX_{I,:}^T \left( D_{I,I} \left( X_{I,:} \boldsymbol{s} \right) \right). \tag{13}$$

## Appendix E. Multi-class SVM by Crammer and Singer

Keerthi et al. (2008) extend the coordinate descent method to a sequential dual method for a multi-class SVM formulation by Crammer and Singer. However, our implementation is slightly different from the one in Keerthi et al. (2008). In the following sections, we describe the formulation and the implementation details, including the stopping condition (Appendix E.4) and the shrinking strategy (Appendix E.5).

### E.1 Formulations

Given a set of instance-label pairs $(\boldsymbol{x}_i, y_i), i = 1, \ldots, l, \boldsymbol{x}_i \in R^n, y_i \in \{1, \ldots, k\}$, Crammer and Singer (2000) proposed a multi-class approach by solving the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w}_m, \xi_i} \quad & \frac{1}{2} \sum_{m=1}^{k} \boldsymbol{w}_m^T \boldsymbol{w}_m + C \sum_{i=1}^{l} \xi_i \\ \text{subject to} \quad & \boldsymbol{w}_{y_i}^T \boldsymbol{x}_i - \boldsymbol{w}_m^T \boldsymbol{x}_i \geq e_i^m - \xi_i, \ i = 1, \ldots, l, \end{aligned} \tag{14}$$

where

$$e_i^m = \begin{cases} 0 & \text{if } y_i = m, \\ 1 & \text{if } y_i \neq m. \end{cases}$$

The decision function is

$$\arg\max_{m=1,\ldots,k} \boldsymbol{w}_m^T \boldsymbol{x}.$$

The dual of (14) is:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{m=1}^{k}\|\boldsymbol{w}_m\|^2 + \sum_{i=1}^{l}\sum_{m=1}^{k} e_i^m \alpha_i^m \\ \text{subject to} \quad & \sum_{m=1}^{k} \alpha_i^m = 0, \forall i = 1,\ldots,l \\ & \alpha_i^m \le C_{y_i}^m, \forall i = 1,\ldots,l, m = 1,\ldots,k, \end{aligned} \tag{15}$$

where

$$\boldsymbol{w}_m = \sum_{i=1}^{l} \alpha_i^m \boldsymbol{x}_i, \forall m, \quad \boldsymbol{\alpha} = [\alpha_1^1,\ldots,\alpha_1^k,\ldots,\alpha_l^1,\ldots,\alpha_l^k]^T. \tag{16}$$

and

$$C_{y_i}^m = \begin{cases} 0 & \text{if } y_i \ne m, \\ C & \text{if } y_i = m. \end{cases} \tag{17}$$

Recently, Keerthi et al. (2008) proposed a sequential dual method to efficiently solve (15). Our implementation is based on this paper. The main differences are the sub-problem solver and the shrinking strategy.

### E.2 The Sequential Dual Method for (15)

The optimization problem (15) has $kl$ variables, which are very large. Therefore, we extend the coordinate descent method to decomposes $\boldsymbol{\alpha}$ into blocks $[\bar{\boldsymbol{\alpha}}_1,\ldots,\bar{\boldsymbol{\alpha}}_l]$, where

$$\bar{\boldsymbol{\alpha}}_i = [\alpha_i^1,\ldots,\alpha_i^k]^T, i = 1,\ldots,l.$$

Each time we select an index $i$ and aim at minimizing the following sub-problem formed by $\bar{\boldsymbol{\alpha}}_i$:

$$\begin{aligned} \min_{\bar{\boldsymbol{\alpha}}_i} \quad & \sum_{m=1}^{k} \frac{1}{2} A(\alpha_i^m)^2 + B_m \alpha_i^m \\ \text{subject to} \quad & \sum_{m=1}^{k} \alpha_i^m = 0, \\ & \alpha_i^m \le C_{y_i}^m, m = \{1,\ldots,k\}, \end{aligned}$$

where

$$A = \boldsymbol{x}_i^T \boldsymbol{x}_i \text{ and } B_m = \boldsymbol{w}_m^T \boldsymbol{x}_i + e_i^m - A\alpha_i^m. \tag{18}$$

In (18), $A$ and $B_m$ are constants obtained using $\boldsymbol{\alpha}$ of the previous iteration..

**Algorithm 1** The coordinate descent method for (15)

- Given $\boldsymbol{\alpha}$ and the corresponding $\boldsymbol{w}_m$
- While $\boldsymbol{\alpha}$ is not optimal, (outer iteration)
  1. Randomly permute $\{1,\ldots,l\}$ to $\{\pi(1),\ldots,\pi(l)\}$
  2. For $i = \pi(1),\ldots,\pi(l)$, (inner iteration)

     If $\bar{\boldsymbol{\alpha}}_i$ is active and $\boldsymbol{x}_i^T\boldsymbol{x}_i \neq 0$ (i.e., $A \neq 0$)
     - Solve a $|U_i|$-variable sub-problem (19)
     - Maintain $\boldsymbol{w}_m$ for all $m$ by (20)

Since bounded variables (i.e., $\alpha_i^m = C_{y_i}^m, \forall m \notin U_i$) can be shrunken during training, we minimize with respect to a sub-vector $\bar{\boldsymbol{\alpha}}_i^{U_i}$, where $U_i \subset \{1,\ldots,k\}$ is an index set. That is, we solve the following $|U_i|$-variable sub-problem while fixing other variables:

$$\begin{aligned} \min_{\bar{\boldsymbol{\alpha}}_i^{U_i}} \quad & \sum_{m\in U_i} \frac{1}{2}A(\alpha_i^m)^2 + B_m\alpha_i^m \\ \text{subject to} \quad & \sum_{m\in U_i} \alpha_i^m = -\sum_{m\notin U_i} \alpha_i^m, \\ & \alpha_i^m \le C_{y_i}^m, m \in U_i. \end{aligned} \tag{19}$$

Notice that there are two situations that we do not solve the sub-problem of index $i$. First, if $|U_i| < 2$, then the whole $\bar{\boldsymbol{\alpha}}_i$ is fixed by the equality constraint in (19). So we can shrink the whole vector $\bar{\boldsymbol{\alpha}}_i$ while training. Second, if $A = 0$, then $\boldsymbol{x}_i = \boldsymbol{0}$ and (16) shows that the value of $\alpha_i^m$ does not affect $\boldsymbol{w}_m$ for all $m$. So the value of $\bar{\boldsymbol{\alpha}}_i$ is independent of other variables and does not affect the final model. Thus we do not need to solve $\bar{\boldsymbol{\alpha}}_i$ for those $\boldsymbol{x}_i = \boldsymbol{0}$.

Similar to Hsieh et al. (2008), we consider a random permutation heuristic. That is, instead of solving sub-problems in the order of $\bar{\boldsymbol{\alpha}}_1,\ldots\bar{\boldsymbol{\alpha}}_l$, we permute $\{1,\ldots l\}$ to $\{\pi(1),\ldots\pi(l)\}$, and solve sub-problems in the order of $\bar{\boldsymbol{\alpha}}_{\pi(1)},\ldots,\bar{\boldsymbol{\alpha}}_{\pi(l)}$. Past results show that such a heuristic gives faster convergence.

We discuss our sub-problem solver in Appendix E.3. After solving the sub-problem, if $\hat{\alpha}_i^m$ is the old value and $\alpha_i^m$ is the value after updating, we maintain $\boldsymbol{w}_m$, defined in (16), by

$$\boldsymbol{w}_m \leftarrow \boldsymbol{w}_m + (\alpha_i^m - \hat{\alpha}_i^m)y_i\boldsymbol{x}_i. \tag{20}$$

To save the computational time, we collect elements satisfying $\alpha_i^m \neq \hat{\alpha}_i^m$ before doing (20). The procedure is described in Algorithm 1.

### E.3 Solving the sub-problem (19)

We adopt the algorithm in Crammer and Singer (2000) but use a rather simple way as in Lee and Lin (2013) to illustrate it. The KKT conditions of (19) indicate that there are

**Algorithm 2** Solving the sub-problem

- Given $A$, $\boldsymbol{B}$
- Compute $\boldsymbol{D}$ by (27)
- Sort $\boldsymbol{D}$ in decreasing order; assume $\boldsymbol{D}$ has elements $D_1, D_2, \ldots, D_{|U_i|}$
- $r \leftarrow 2, \beta \leftarrow D_1 - AC$
- While $r \leq |U_i|$ and $\beta/(r-1) < D_r$
  1. $\beta \leftarrow \beta + D_r$
  2. $r \leftarrow r + 1$
- $\beta \leftarrow \beta/(r-1)$
- $\alpha_i^m \leftarrow \min(C_{y_i}^m, (\beta - B_m)/A), \forall m$

scalars $\beta, \rho_m, m \in U_i$ such that

$$\sum_{m \in U_i} \alpha_i^m = -\sum_{m \notin U_i} C_{y_i}^m, \tag{21}$$

$$\alpha_i^m \leq C_{y_i}^m, \forall m \in U_i, \tag{22}$$

$$\rho_m(C_{y_i}^m - \alpha_i^m) = 0, \rho_m \geq 0, \forall m \in U_i, \tag{23}$$

$$A\alpha_i^m + B_m - \beta = -\rho_m, \forall m \in U_i. \tag{24}$$

Using (22), equations (23) and (24) are equivalent to

$$A\alpha_i^m + B_m - \beta = 0, \text{ if } \alpha_i^m < C_{y_i}^m, \forall m \in U_i, \tag{25}$$

$$A\alpha_i^m + B_m - \beta = AC_{y_i}^m + B_m - \beta \leq 0, \text{ if } \alpha_i^m = C_{y_i}^m, \forall m \in U_i. \tag{26}$$

Now KKT conditions become (21)-(22), and (25)-(26). For simplicity, we define

$$D_m = B_m + AC_{y_i}^m, m = 1, \ldots, k. \tag{27}$$

If $\beta$ is known, then we can show that

$$\alpha_i^m \equiv \min(C_{y_i}^m, \frac{\beta - B_m}{A}) \tag{28}$$

satisfies all KKT conditions except (21). Clearly, the way to get $\alpha_i^m$ in (28) gives $\alpha_i^m \leq C_{y_i}^m, \forall m \in U_i$, so (22) holds. From (28), when

$$\frac{\beta - B_m}{A} < C_{y_i}^m, \text{ which is equivalent to } \beta < D_m, \tag{29}$$

we have

$$\alpha_i^m = \frac{\beta - B_m}{A} < C_{y_i}^m, \text{ which implies } \beta - B_m = A\alpha_i^m.$$

Thus, (25) is satisfied. Otherwise, $\beta \geq D_m$ and $\alpha_i^m = C_{y_i}^m$ satisfies (26).

The remaining task is how to find $\beta$ such that (21) holds. From (21) and (25) we obtain

$$\sum_{\substack{m:m\in U_i\\ \alpha_i^m<C_{y_i}^m}} (\beta - B_m) = -(\sum_{m:m\notin U_i} AC_{y_i}^m + \sum_{\substack{m:m\in U_i\\ \alpha=C_{y_i}^m}} AC_{y_i}^m).$$

Then,

$$\begin{aligned}\sum_{\substack{m:m\in U_i\\ \alpha_i^m<C_{y_i}^m}} \beta &= \sum_{\substack{m:m\in U_i\\ \alpha_i^m<C_{y_i}^m}} D_m - \sum_{m=1}^{k} AC_{y_i}^m \\ &= \sum_{\substack{m:m\in U_i\\ \alpha_i^m<C_{y_i}^m}} D_m - AC.\end{aligned}$$

Hence,

$$\beta = \frac{\sum_{m\in U_i, \alpha_i^m<C_{y_i}^m} D_m - AC}{|\{m \mid m \in U_i, \alpha_i^m < C_{y_i}^m\}|}. \tag{30}$$

Combining (30) and (26), we begin with a set $\Phi = \phi$, and then sequentially add one index $m$ to $\Phi$ by the decreasing order of $D_m$, $m = 1, \ldots, k, m \neq y_i$ until

$$h = \frac{\sum_{m\in\Phi} D_m - AC}{|\Phi|} \geq \max_{m\notin\Phi} D_m. \tag{31}$$

Let $\beta = h$ when (31) is satisfied. Algorithm 2 lists the details for solving the sub-problem (19). To prove (21), it is sufficient to show that $\beta$ and $\alpha_i^m, \forall m \in U_i$ obtained by Algorithm 2 satisfy (30). This is equivalent to showing that the final $\Phi$ satisfies

$$\Phi = \{m \mid m \in U_i, \alpha_i^m < C_{y_i}^m\}.$$

From (28) and (29), we prove the following equivalent result.

$$\beta < D_m, \forall m \in \Phi \text{ and } \beta \geq D_m, \forall m \notin \Phi. \tag{32}$$

The second inequality immediately follows from (31). For the first, assume $t$ is the last element added to $\Phi$. When it is considered, (31) is not satisfied yet, so

$$\frac{\sum_{m\in\Phi\setminus\{t\}} D_m - AC}{|\Phi| - 1} < D_t. \tag{33}$$

Using (33) and the fact that elements in $\Phi$ are added in the decreasing order of $D_m$,

$$\begin{aligned}\sum_{m\in\Phi} D_m - AC &= \sum_{m\in\Phi\setminus\{t\}} D_m + D_t - AC \\ &< (|\Phi| - 1)D_t + D_t = |\Phi|D_t \\ &\leq |\Phi|D_s, \qquad \forall s \in \Phi.\end{aligned}$$

Thus, we have the first inequality in (32).

With all KKT conditions satisfied, Algorithm 2 obtains an optimal solution of (19).

### E.4 Stopping Condition

The KKT optimality conditions of (15) imply that there are $b_1, \dots, b_l \in R$ such that for all $i = 1, \dots, l$, $m = 1, \dots, k$,

$$\begin{aligned} \boldsymbol{w}_m^T \boldsymbol{x}_i + e_i^m - b_i = 0 \text{ if } \alpha_i^m < C_i^m, \\ \boldsymbol{w}_m^T \boldsymbol{x}_i + e_i^m - b_i \le 0 \text{ if } \alpha_i^m = C_i^m. \end{aligned}$$

Let

$$G_i^m = \frac{\partial f(\boldsymbol{\alpha})}{\partial \alpha_i^m} = \boldsymbol{w}_m^T \boldsymbol{x}_i + e_i^m, \forall i, m,$$

the optimality of $\boldsymbol{\alpha}$ holds if and only if

$$\max_m G_i^m - \min_{m:\alpha_i^m < C_i^m} G_i^m = 0, \forall i. \tag{34}$$

At each inner iteration, we first compute $G_i^m$ and define:

$$\text{minG} \equiv \min_{m:\alpha_i^m < C_i^m} G_i^m, \text{maxG} \equiv \max_m G_i^m, S_i = \text{maxG} - \text{minG}.$$

Then the stopping condition for a tolerance $\epsilon > 0$ can be checked by

$$\max_i S_i < \epsilon. \tag{35}$$

Note that maxG and minG are calculated based on the latest $\boldsymbol{\alpha}$ (i.e., $\boldsymbol{\alpha}$ after each inner iteration).

### E.5 Shrinking Strategy

The shrinking technique reduces the size of the problem without considering some bounded variables. Eq. (34) suggests that we can shrink $\alpha_i^m$ out if $\alpha_i^m$ satisfies the following condition:

$$\alpha_i^m = C_{y_i}^m \text{ and } G_i^m < \text{minG}. \tag{36}$$

Then we solve a $|U_i|$-variable sub-problem (19). To implement this strategy, we maintain an $l \times k$ index array `alpha_index` and an $l$ array `activesize_i`, such that `activesize_i[i]` $= |U_i|$. We let the first `activesize_i[i]` elements in `alpha_index[i]` are active indices, and others are shrunken indices. Moreover, we need to maintain an $l$-variable array `y_index`, such that

$$\texttt{alphaindex[i][y\_index[i]]} = y_i. \tag{37}$$

When we shrink a index `alpha_index[i][m]` out, we first find the largest $\bar{m}$ such that $\bar{m} <$ `activesize_i[i]` and `alpha_index[i]`$[\bar{\text{m}}]$ does not satisfy the shrinking condition (36), then swap the two elements and decrease `activesize_i[i]` by 1. Note that if we swap index $y_i$, we need to maintain `y_index[i]` to ensure (37). For the instance level shrinking and random permutation, we also maintain a index array `index` and a variable `activesize` similar to `alpha_index` and `activesize_i`, respectively. We let the first `activesize` elements of `index` be active indices, while others be shrunken indices. When $|U_i|$, the active size of $\bar{\boldsymbol{\alpha}}_i$,

**Algorithm 3** Shrinking strategy

- Given $\epsilon$
- Begin with $\epsilon_{\text{shrink}} \leftarrow \max(1, 10\epsilon)$, `start_from_all` $\leftarrow True$
- While
  1. For all active $\bar{\boldsymbol{\alpha}}_i$
     (a) Do shrinking and calculate $S_i$
     (b) `stopping` $\leftarrow \max($`stopping`$, S_i)$
     (c) Optimize over active variables in $\bar{\boldsymbol{\alpha}}_i$
  2. If `stopping` $< \epsilon_{\text{shrink}}$
     (a) If `stopping` $< \epsilon$ and `start_from_all` is $True$, BREAK
     (b) Take all shrunken variables back
     (c) `start_from_all` $\leftarrow True$
     (d) $\epsilon_{\text{shrink}} \leftarrow \max(\epsilon, \epsilon_{\text{shrink}}/2)$

     Else
     (a) `start_from_all` $\leftarrow False$

is less than 2 (`activesize_i[i]` $< 2$), we swap this index with the last active element in `index`, and decrease `activesize` by 1.

However, experiments show that (36) is too aggressive. There are too many wrongly shrunken variables. To deal with this problem, we use an $\epsilon$-cooling strategy. Given a pre-specified stopping tolerance $\epsilon$, we begin with

$$\epsilon_{\text{shrink}} = \max(1, 10\epsilon)$$

and decrease it by a factor of 2 in each graded step until $\epsilon_{\text{shrink}} \leq \epsilon$.

The program ends if the stopping condition (35) is satisfied. But we can exactly compute (35) only when there are no shrunken variables. Thus the process stops under the following two conditions:

1. None of the instances is shrunken in the beginning of the loop.
2. (35) is satisfied.

Our shrinking strategy is in Algorithm 3.

Regarding the random permutation, we permute the first `activesize` elements of `index` at each outer iteration, and then sequentially solve the sub-problems.

## Appendix F. L1-regularized L2-loss Support Vector Machines

In this section, we describe details of a coordinate descent method for L1-regularized L2-loss support vector machines. The problem formulation is in (7). Our procedure is similar to

Chang et al. (2008) for L2-regularized L2-loss SVM, but we make certain modifications to handle the non-differentiability due to the L1 regularization. It is also related to Tseng and Yun (2009). See detailed discussions of theoretical properties in Yuan et al. (2010).

Define

$$b_i(\boldsymbol{w}) \equiv 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i \quad \text{and} \quad I(\boldsymbol{w}) \equiv \{i \mid b_i(\boldsymbol{w}) > 0\}.$$

The one-variable sub-problem for the $j$th variable is a function of $z$:

$$\begin{aligned} & f(\boldsymbol{w} + z\boldsymbol{e}_j) - f(\boldsymbol{w}) \\ &= |w_j + z| - |w_j| + C \sum_{i \in I(\boldsymbol{w} + z\boldsymbol{e}_j)} b_i(\boldsymbol{w} + z\boldsymbol{e}_j)^2 - C \sum_{i \in I(\boldsymbol{w})} b_i(\boldsymbol{w})^2 \\ &= |w_j + z| + L_j(z; \boldsymbol{w}) + \text{constant} \\ &\approx |w_j + z| + L'_j(0; \boldsymbol{w}) z + \frac{1}{2} L''_j(0; \boldsymbol{w}) z^2 + \text{constant}, \end{aligned} \tag{38}$$

where

$$\boldsymbol{e}_j = [\underbrace{0, \ldots, 0}_{j-1}, 1, 0, \ldots, 0]^T \in R^n, \tag{39}$$

$$\begin{aligned} L_j(z; \boldsymbol{w}) &\equiv C \sum_{i \in I(\boldsymbol{w} + z\boldsymbol{e}_j)} b_i(\boldsymbol{w} + z\boldsymbol{e}_j)^2, \\ L'_j(0; \boldsymbol{w}) &= -2C \sum_{i \in I(\boldsymbol{w})} y_i x_{ij} b_i(\boldsymbol{w}), \end{aligned}$$

and

$$L''_j(0; \boldsymbol{w}) = \max(2C \sum_{i \in I(\boldsymbol{w})} x_{ij}^2,\ 10^{-12}). \tag{40}$$

Note that $L_j(z; \boldsymbol{w})$ is differentiable but not twice differentiable, so $2C \sum_{i \in I(\boldsymbol{w})} x_{ij}^2$ in (40) is a generalized second derivative (Chang et al., 2008) at $z = 0$. This value may be zero if $x_{ij} = 0,\ \forall i \in I(\boldsymbol{w})$, so we further make it strictly positive. The Newton direction from minimizing (38) is

$$d = \begin{cases} -\frac{L'_j(0;\boldsymbol{w})+1}{L''_j(0;\boldsymbol{w})} & \text{if } L'_j(0; \boldsymbol{w}) + 1 \le L''_j(0; \boldsymbol{w}) w_j, \\ -\frac{L'_j(0;\boldsymbol{w})-1}{L''_j(0;\boldsymbol{w})} & \text{if } L'_j(0; \boldsymbol{w}) - 1 \ge L''_j(0; \boldsymbol{w}) w_j, \\ -w_j & \text{otherwise.} \end{cases}$$

See the derivation in (Yuan et al., 2010, Appendix B). We then conduct a line search procedure to check if $d$, $\beta d$, $\beta^2 d$, ..., satisfy the following sufficient decrease condition:

$$\begin{aligned} & |w_j + \beta^t d| - |w_j| + C \sum_{i \in I(\boldsymbol{w} + \beta^t d\boldsymbol{e}_j)} b_i(\boldsymbol{w} + \beta^t d\boldsymbol{e}_j)^2 - C \sum_{i \in I(\boldsymbol{w})} b_i(\boldsymbol{w})^2 \\ &\le \sigma \beta^t \left( L'_j(0; \boldsymbol{w}) d + |w_j + d| - |w_j| \right), \end{aligned} \tag{41}$$

where $t = 0, 1, 2, \ldots$, $\beta \in (0, 1)$, and $\sigma \in (0, 1)$. From Chang et al. (2008, Lemma 5),

$$C \sum_{i \in I(\boldsymbol{w}+d\boldsymbol{e}_j)} b_i(\boldsymbol{w} + d\boldsymbol{e}_j)^2 - C \sum_{i \in I(\boldsymbol{w})} b_i(\boldsymbol{w})^2 \leq C(\sum_{i=1}^{l} x_{ij}^2)d^2 + L_j'(0; \boldsymbol{w})d.$$

We can precompute $\sum_{i=1}^{l} x_{ij}^2$ and check

$$\begin{aligned} &|w_j + \beta^t d| - |w_j| + C(\sum_{i=1}^{l} x_{ij}^2)(\beta^t d)^2 + L_j'(0; \boldsymbol{w})\beta^t d \\ &\leq \sigma\beta^t \left( L_j'(0; \boldsymbol{w})d + |w_j + d| - |w_j| \right), \end{aligned} \tag{42}$$

before (41). Note that checking (42) is very cheap. The main cost in checking (41) is on calculating $b_i(\boldsymbol{w} + \beta^t d\boldsymbol{e}_j),\ \forall i$. To save the number of operations, if $b_i(\boldsymbol{w})$ is available, one can use

$$b_i(\boldsymbol{w} + \beta^t d\boldsymbol{e}_j) = b_i(\boldsymbol{w}) - (\beta^t d) y_i x_{ij}. \tag{43}$$

Therefore, we store and maintain $b_i(\boldsymbol{w})$ in an array. Since $b_i(\boldsymbol{w})$ is used in every line search step, we cannot override its contents. After the line search procedure, we must run (43) again to update $b_i(\boldsymbol{w})$. That is, the same operation (43) is run twice, where the first is for checking the sufficient decrease condition and the second is for updating $b_i(\boldsymbol{w})$. Alternatively, one can use another array to store $b_i(\boldsymbol{w} + \beta^t d\boldsymbol{e}_j)$ and copy its contents to the array for $b_i(\boldsymbol{w})$ in the end of the line search procedure. We propose the following trick to use only one array and avoid the duplicated computation of (43). From

$$\begin{aligned} b_i(\boldsymbol{w} + d\boldsymbol{e}_j) &= b_i(\boldsymbol{w}) - d y_i x_{ij}, \\ b_i(\boldsymbol{w} + \beta d\boldsymbol{e}_j) &= b_i(\boldsymbol{w} + d\boldsymbol{e}_j) + (d - \beta d) y_i x_{ij}, \\ b_i(\boldsymbol{w} + \beta^2 d\boldsymbol{e}_j) &= b_i(\boldsymbol{w} + \beta d\boldsymbol{e}_j) + (\beta d - \beta^2 d) y_i x_{ij}, \\ &\vdots \end{aligned} \tag{44}$$

at each line search step, we obtain $b_i(\boldsymbol{w} + \beta^t d\boldsymbol{e}_j)$ from $b_i(\boldsymbol{w} + \beta^{t-1} d\boldsymbol{e}_j)$ in order to check the sufficient decrease condition (41). If the condition is satisfied, then the $b_i$ array already has values needed for the next sub-problem. If the condition is not satisfied, using $b_i(\boldsymbol{w} + \beta^t d\boldsymbol{e}_j)$ on the right-hand side of the equality (44), we can obtain $b_i(\boldsymbol{w} + \beta^{t+1} d\boldsymbol{e}_j)$ for the next line search step. Therefore, we can simultaneously check the sufficient decrease condition and update the $b_i$ array. A summary of the procedure is in Algorithm 4.

The stopping condition is by checking the optimality condition. An optimal $w_j$ satisfies

$$\begin{cases} L_j'(0; \boldsymbol{w}) + 1 = 0 & \text{if } w_j > 0, \\ L_j'(0; \boldsymbol{w}) - 1 = 0 & \text{if } w_j < 0, \\ -1 \leq L_j'(0; \boldsymbol{w}) \leq 1 & \text{if } w_j = 0. \end{cases} \tag{45}$$

We calculate

$$v_j \equiv \begin{cases} |L_j'(0; \boldsymbol{w}) + 1| & \text{if } w_j > 0, \\ |L_j'(0; \boldsymbol{w}) - 1| & \text{if } w_j < 0, \\ \max\left( L_j'(0; \boldsymbol{w}) - 1,\ -1 - L_j'(0; \boldsymbol{w}),\ 0 \right) & \text{if } w_j = 0, \end{cases}$$

to measure how the optimality condition is violated. The procedure stops if

$$\sum_{j=1}^{n} (v_j \text{ at the current iteration})$$
$$\leq 0.01 \times \frac{\min(\#\text{pos}, \#\text{neg})}{l} \times \sum_{j=1}^{n} (v_j \text{ at the 1st iteration}),$$

where #pos and #neg indicate the numbers of positive and negative labels in a data set, respectively.

Due to the sparsity of the optimal solution, some $w_j$ become zeros in the middle of the optimization procedure and are not changed subsequently. We can shrink these $w_j$ components to reduce the number of variables. From (45), an optimal $w_j$ satisfies that

$$-1 < L'_j(0; \boldsymbol{w}) < 1 \quad \text{implies} \quad w_j = 0.$$

If at one iteration, $w_j = 0$ and

$$-1 + M \leq L'_j(0; \boldsymbol{w}) \leq 1 - M,$$

where

$$M \equiv \frac{\max_j (v_j \text{ at the previous iteration})}{l},$$

we conjecture that $w_j$ will not be changed in subsequent iterations. We then remove this $w_j$ from the optimization problem.

## Appendix G. L1-regularized Logistic Regression

In LIBLINEAR (after version 1.8), we implement an improved GLMNET, called newGLMNET, for solving L1-regularized logistic regression. GLMNET was proposed by Friedman et al. (2010), while details of newGLMNET can be found in Yuan et al. (2012). Here, we provide implementation details not mentioned in Yuan et al. (2012).

The problem formulation is in (8). To avoid handling $y_i$ in $e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}$, we reformulate $f(\boldsymbol{w})$ as

$$f(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 + C \left( \sum_{i=1}^{l} \log(1 + e^{-\boldsymbol{w}^T \boldsymbol{x}_i}) + \sum_{i:y_i=-1} \boldsymbol{w}^T \boldsymbol{x}_i \right).$$

We define

$$L(\boldsymbol{w}) \equiv C \left( \sum_{i=1}^{l} \log(1 + e^{-\boldsymbol{w}^T \boldsymbol{x}_i}) + \sum_{i:y_i=-1} \boldsymbol{w}^T \boldsymbol{x}_i \right).$$

The gradient and Hessian of $\nabla L(\boldsymbol{w})$ can be written as

$$\begin{aligned} \nabla_j L(\boldsymbol{w}) &= C \left( \sum_{i=1}^{l} \frac{-x_{ij}}{1 + e^{\boldsymbol{w}^T \boldsymbol{x}_i}} + \sum_{i:y_i=-1} x_{ij} \right), \text{ and} \\ \nabla^2_{jj} L(\boldsymbol{w}) &= C \left( \sum_{i=1}^{l} \left( \frac{x_{ij}}{1 + e^{\boldsymbol{w}^T \boldsymbol{x}_i}} \right)^2 e^{\boldsymbol{w}^T \boldsymbol{x}_i} \right). \end{aligned} \tag{46}$$

For line search, we use the following form of the sufficient decrease condition:

$$\begin{aligned}
&f(\boldsymbol{w}+\beta^t\boldsymbol{d}) - f(\boldsymbol{w}) \\
&= \|\boldsymbol{w}+\beta^t\boldsymbol{d}\|_1 - \|\boldsymbol{w}\|_1 + C\left(\sum_{i=1}^{l}\log\left(\frac{1+e^{-(\boldsymbol{w}+\beta^t\boldsymbol{d})^T\boldsymbol{x}_i}}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}_i}}\right) + \beta^t\sum_{i:y_i=-1}\boldsymbol{d}^T\boldsymbol{x}_i\right) \\
&= \|\boldsymbol{w}+\beta^t\boldsymbol{d}\|_1 - \|\boldsymbol{w}\|_1 + C\left(\sum_{i=1}^{l}\log\left(\frac{e^{(\boldsymbol{w}+\beta^t\boldsymbol{d})^T\boldsymbol{x}_i}+1}{e^{(\boldsymbol{w}+\beta^t\boldsymbol{d})^T\boldsymbol{x}_i}+e^{\beta^t\boldsymbol{d}^T\boldsymbol{x}_i}}\right) + \beta^t\sum_{i:y_i=-1}\boldsymbol{d}^T\boldsymbol{x}_i\right) \\
&\le \sigma\beta^t\left(\nabla L(\boldsymbol{w})^T\boldsymbol{d} + \|\boldsymbol{w}+\boldsymbol{d}\|_1 - \|\boldsymbol{w}\|_1\right), \qquad (47)
\end{aligned}$$

where $\boldsymbol{d}$ is the search direction, $\beta \in (0,1)$ and $\sigma \in (0,1)$. From (46) and (47), all we need is to maintain $\boldsymbol{d}^T\boldsymbol{x}_i$ and $e^{\boldsymbol{w}^T\boldsymbol{x}_i}, \forall i$. We update $e^{\boldsymbol{w}^T\boldsymbol{x}_i}$ by

$$e^{(\boldsymbol{w}+\lambda\boldsymbol{d})^T\boldsymbol{x}_i} = e^{\boldsymbol{w}^T\boldsymbol{x}_i}\cdot e^{\lambda\boldsymbol{d}^T\boldsymbol{x}_i}, \quad \forall i.$$

## Appendix H. Implementation of L1-regularized Logistic Regression in LIBLINEAR Versions 1.4–1.7

In the earlier versions of LIBLINEAR (versions 1.4–1.7), a coordinate descent method is implemented for L1-regularized logistic regression. It is similar to the method for L1-regularized L2-loss SVM in Appendix F.

The problem formulation is in (8). To avoid handling $y_i$ in $e^{-y_i\boldsymbol{w}^T\boldsymbol{x}_i}$, we reformulate $f(\boldsymbol{w})$ as

$$f(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 + C\left(\sum_{i=1}^{l}\log(1+e^{-\boldsymbol{w}^T\boldsymbol{x}_i}) + \sum_{i:y_i=-1}\boldsymbol{w}^T\boldsymbol{x}_i\right).$$

At each iteration, we select an index $j$ and minimize the following one-variable function of $z$:

$$\begin{aligned}
&f(\boldsymbol{w}+z\boldsymbol{e}_j) - f(\boldsymbol{w}) \\
&= |w_j+z| - |w_j| + C\left(\sum_{i=1}^{l}\log(1+e^{-(\boldsymbol{w}+z\boldsymbol{e}_j)^T\boldsymbol{x}_i}) + \sum_{i:y_i=-1}(\boldsymbol{w}+z\boldsymbol{e}_j)^T\boldsymbol{x}_i\right) \\
&\quad - C\left(\sum_{i=1}^{l}\log(1+e^{-\boldsymbol{w}^T\boldsymbol{x}_i}) + \sum_{i:y_i=-1}\boldsymbol{w}^T\boldsymbol{x}_i\right) \\
&= |w_j+z| + L_j(z;\boldsymbol{w}) + \text{constant} \\
&\approx |w_j+z| + L_j'(0;\boldsymbol{w})z + \frac{1}{2}L_j''(0;\boldsymbol{w})z^2 + \text{constant}, \qquad (48)
\end{aligned}$$

where $\boldsymbol{e}_j$ is defined in (39),

$$L_j(z;\boldsymbol{w}) \equiv C\left(\sum_{i=1}^{l}\log(1+e^{-(\boldsymbol{w}+z\boldsymbol{e}_j)^T\boldsymbol{x}_i}) + \sum_{i:y_i=-1}(\boldsymbol{w}+z\boldsymbol{e}_j)^T\boldsymbol{x}_i\right),$$
$$L_j'(0;\boldsymbol{w}) = C\left(\sum_{i=1}^{l}\frac{-x_{ij}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1} + \sum_{i:y_i=-1}x_{ij}\right), \text{ and}$$
$$L_j''(0;\boldsymbol{w}) = C\left(\sum_{i=1}^{l}\left(\frac{x_{ij}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}\right)^2 e^{\boldsymbol{w}^T\boldsymbol{x}_i}\right).$$

The optimal solution $d$ of minimizing (48) is:

$$d = \begin{cases} -\frac{L_j'(0;\boldsymbol{w})+1}{L_j''(0;\boldsymbol{w})} & \text{if } L_j'(0;\boldsymbol{w})+1 \le L_j''(0;\boldsymbol{w})w_j, \\ -\frac{L_j'(0;\boldsymbol{w})-1}{L_j''(0;\boldsymbol{w})} & \text{if } L_j'(0;\boldsymbol{w})-1 \ge L_j''(0;\boldsymbol{w})w_j, \\ -w_j & \text{otherwise.} \end{cases}$$

We then conduct a line search procedure to check if $d$, $\beta d$, $\beta^2 d$, ..., satisfy the following sufficient decrease condition:

$$\begin{aligned} & f(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j) - f(\boldsymbol{w}) \\ &= C\left(\sum_{i:x_{ij}\neq 0}\log\left(\frac{1+e^{-(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^T\boldsymbol{x}_i}}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}_i}}\right) + \beta^t d\sum_{i:y_i=-1}x_{ij}\right) + |w_j+\beta^t d| - |w_j| \\ &= \sum_{i:x_{ij}\neq 0}^{l} C\log\left(\frac{e^{(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^T\boldsymbol{x}_i}+1}{e^{(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^T\boldsymbol{x}_i}+e^{\beta^t d x_{ij}}}\right) + \beta^t d\sum_{i:y_i=-1}Cx_{ij} + |w_j+\beta^t d| - |w_j| \qquad (49) \\ &\le \sigma\beta^t\left(L_j'(0;\boldsymbol{w})d + |w_j+d| - |w_j|\right), \qquad (50) \end{aligned}$$

where $t = 0, 1, 2, \ldots$, $\beta \in (0,1)$, and $\sigma \in (0,1)$.

As the computation of (49) is expensive, similar to (42) for L2-loss SVM, we derive an upper bound for (49). If

$$x_j^* \equiv \max_i x_{ij} \ge 0,$$

then

$$\sum_{i:x_{ij}\neq 0} C\log\left(\frac{e^{(\boldsymbol{w}+d\boldsymbol{e}_j)^T\boldsymbol{x}_i}+1}{e^{(\boldsymbol{w}+d\boldsymbol{e}_j)^T\boldsymbol{x}_i}+e^{dx_{ij}}}\right) = \sum_{i:x_{ij}\neq 0} C\log\left(\frac{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+e^{-dx_{ij}}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}\right)$$

$$\leq (\sum_{i:x_{ij}\neq 0} C)\log\left(\frac{\sum_{i:x_{ij}\neq 0} C\frac{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+e^{-dx_{ij}}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}}{\sum_{i:x_{ij}\neq 0} C}\right) \tag{51}$$

$$= (\sum_{i:x_{ij}\neq 0} C)\log\left(1+\frac{\sum_{i:x_{ij}\neq 0} C\left(\frac{1}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}(e^{-dx_{ij}}-1)\right)}{\sum_{i:x_{ij}\neq 0} C}\right)$$

$$\leq (\sum_{i:x_{ij}\neq 0} C)\log\left(1+\frac{\sum_{i:x_{ij}\neq 0} C\left(\frac{1}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}(\frac{x_{ij}e^{-dx_j^*}}{x_j^*}+\frac{x_j^*-x_{ij}}{x_j^*}-1)\right)}{\sum_{i:x_{ij}\neq 0} C}\right) \tag{52}$$

$$= (\sum_{i:x_{ij}\neq 0} C)\log\left(1+\frac{(e^{-dx_j^*}-1)\sum_{i:x_{ij}\neq 0}\frac{Cx_{ij}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}}{x_j^*\sum_{i:x_{ij}\neq 0} C}\right), \tag{53}$$

where (51) is from Jensen's inequality and (52) is from the convexity of the exponential function:

$$e^{-dx_{ij}} \leq \frac{x_{ij}}{x_j^*}e^{-dx_j^*}+\frac{x_j^*-x_{ij}}{x_j^*}e^0 \quad \text{if } x_{ij}\geq 0. \tag{54}$$

As $f(\boldsymbol{w})$ can also be represented as

$$f(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 + C\left(\sum_{i=1}^{l}\log(1+e^{\boldsymbol{w}^T\boldsymbol{x}_i})-\sum_{i:y_i=1}\boldsymbol{w}^T\boldsymbol{x}_i\right),$$

we can derive another similar bound

$$\sum_{i:x_{ij}\neq 0} C\log\left(\frac{1+e^{(\boldsymbol{w}+d\boldsymbol{e}_j)^T\boldsymbol{x}_i}}{1+e^{\boldsymbol{w}^T\boldsymbol{x}_i}}\right)$$

$$\leq (\sum_{i:x_{ij}\neq 0} C)\log\left(1+\frac{(e^{dx_j^*}-1)\sum_{i:x_{ij}\neq 0}\frac{Cx_{ij}e^{\boldsymbol{w}^T\boldsymbol{x}_i}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}}{x_j^*\sum_{i:x_{ij}\neq 0} C}\right). \tag{55}$$

Therefore, before checking the sufficient decrease condition (50), we check if

$$\begin{aligned}&\min\big((55)-\beta^t d\sum_{i:y_i=1} Cx_{ij}+|w_j+\beta^t d|-|w_j|,\\ &\qquad (53)+\beta^t d\sum_{i:y_i=-1} Cx_{ij}+|w_j+\beta^t d|-|w_j|\big)\\ &\leq \sigma\beta^t\left(L_j'(0;\boldsymbol{w})d+|w_j+d|-|w_j|\right).\end{aligned} \tag{56}$$

Note that checking (56) is very cheap since we already have

$$\sum_{i:x_{ij}\neq 0} \frac{Cx_{ij}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1} \text{ and } \sum_{i:x_{ij}\neq 0} \frac{Cx_{ij}e^{\boldsymbol{w}^T\boldsymbol{x}_i}}{e^{\boldsymbol{w}^T\boldsymbol{x}_i}+1}$$

in calculating $L'_j(0;\boldsymbol{w})$ and $L''_j(0;\boldsymbol{w})$. However, to apply (56) we need that the data set satisfies $x_{ij}\geq 0,\ \forall i,\forall j$. This condition is used in (54).

The main cost in checking (49) is on calculating $e^{(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^T\boldsymbol{x}_i},\ \forall i$. To save the number of operations, if $e^{\boldsymbol{w}^T\boldsymbol{x}_i}$ is available, one can use

$$e^{(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^T\boldsymbol{x}_i} = e^{\boldsymbol{w}^T\boldsymbol{x}_i}e^{(\beta^t d)x_{ij}}.$$

Therefore, we store and maintain $e^{\boldsymbol{w}^T\boldsymbol{x}_i}$ in an array. The setting is similar to the array $1-y_i\boldsymbol{w}^T\boldsymbol{x}_i$ for L2-loss SVM in Appendix F, so one faces the same problem of not being able to check the sufficient decrease condition (50) and update the $e^{\boldsymbol{w}^T\boldsymbol{x}_i}$ array together. We can apply the same trick in Appendix F, but the implementation is more complicated. In our implementation, we allocate another array to store $e^{(\boldsymbol{w}+\beta^t d\boldsymbol{e}_j)^T\boldsymbol{x}_i}$ and copy its contents for updating the $e^{\boldsymbol{w}^T\boldsymbol{x}_i}$ array in the end of the line search procedure. A summary of the procedure is in Algorithm 5.

The stopping condition and the shrinking implementation to remove some $w_j$ components are similar to those for L2-loss support vector machines (see Appendix F).

## Appendix I. L2-regularized Logistic Regression (Solving Dual)

See Yu et al. (2011) for details of a dual coordinate descent method.

## Appendix J. L2-regularized Logistic Regression (Solving Dual)

See Yu et al. (2011) for details of a dual coordinate descent method.

## Appendix K. L2-regularized Support Vector Regression

LIBLINEAR solves both the primal and the dual of L2-loss SVR, but only the dual of L1-loss SVR. The dual SVR is solved by a coordinate descent, and the primal SVR is solved by a trust region Newton method. See Ho and Lin (2012) for details of them. In Appendix C, we mentioned that the trust region Newton method is improved in version 2.11 and 2.20 (See details in Hsia et al. (2017) and Hsia et al. (2018)). Both modifications are applicable to the primal SVR solver.

We mentioned in Appendix C that after version 2.40, the trust-region Newton method is replaced by a line-search Newton method (Galli and Lin, 2020a). This new setting is now used for the primal SVR solver.

## Appendix L. Probability Outputs

Currently we support probability outputs for logistic regression only. Although it is possible to apply techniques in LIBSVM to obtain probabilities for SVM, we decide not to implement it for code simplicity.

The probability model of logistic regression is

$$P(y|\boldsymbol{x}) = \frac{1}{1+e^{-y\boldsymbol{w}^T\boldsymbol{x}}}, \text{ where } y = \pm 1,$$

so probabilities for two-class classification are immediately available. For a $k$-class problem, we need to couple $k$ probabilities

$$\begin{aligned} 1 \text{ vs. not } 1: &\quad P(y=1|\boldsymbol{x}) = 1/(1+e^{-\boldsymbol{w}_1^T\boldsymbol{x}}) \\ &\vdots \\ k \text{ vs. not } k: &\quad P(y=k|\boldsymbol{x}) = 1/(1+e^{-\boldsymbol{w}_k^T\boldsymbol{x}}) \end{aligned}$$

together, where $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k$ are $k$ model vectors obtained after the one-vs-the-rest strategy. Note that the sum of the above $k$ values is not one. A procedure to couple them can be found in Section 4.1 of Huang et al. (2006), but for simplicity, we implement the following heuristic:

$$P(y|\boldsymbol{x}) = \frac{1/(1+e^{-\boldsymbol{w}_y^T\boldsymbol{x}})}{\sum_{m=1}^{k} 1/(1+e^{-\boldsymbol{w}_m^T\boldsymbol{x}})}.$$

## Appendix M. Automatic and Efficient Parameter Selection

See Chu et al. (2015); Hsia and Lin (2020) for details of the procedure and Section N.5 for the practical use.

**Algorithm 4** A coordinate descent algorithm for L1-regularized L2-loss SVC

- Choose $\beta = 0.5$ and $\sigma = 0.01$. Given initial $\boldsymbol{w} \in R^n$.
- Calculate
$$b_i \leftarrow 1 - y_i \boldsymbol{w}^T \boldsymbol{x}_i, \ i = 1, \ldots, l.$$
- While $\boldsymbol{w}$ is not optimal

  For $j = 1, 2, \ldots, n$

  1. Find the Newton direction by solving
$$\min_z \quad |w_j + z| + L'_j(0; \boldsymbol{w})z + \frac{1}{2} L''_j(0; \boldsymbol{w})z^2.$$
  The solution is
$$d = \begin{cases} -\frac{L'_j(0;\boldsymbol{w})+1}{L''_j(0;\boldsymbol{w})} & \text{if } L'_j(0;\boldsymbol{w}) + 1 \le L''_j(0;\boldsymbol{w})w_j, \\ -\frac{L'_j(0;\boldsymbol{w})-1}{L''_j(0;\boldsymbol{w})} & \text{if } L'_j(0;\boldsymbol{w}) - 1 \ge L''_j(0;\boldsymbol{w})w_j, \\ -w_j & \text{otherwise.} \end{cases}$$
  2. $\bar{d} \leftarrow 0$; $\Delta \leftarrow L'_j(0; \boldsymbol{w})d + |w_j + d| - |w_j|$.
  3. While $t = 0, 1, 2, \ldots$
     (a) If
$$|w_j + d| - |w_j| + C(\sum_{i=1}^{l} x_{ij}^2)d^2 + L'_j(0; \boldsymbol{w})d \le \sigma\Delta,$$
     then
$$b_i \leftarrow b_i + (\bar{d} - d)y_i x_{ij}, \ \forall i \text{ and BREAK.}$$
     (b) If $t = 0$, calculate
$$L_{j,0} \leftarrow C \sum_{i \in I(\boldsymbol{w})} b_i^2.$$
     (c) $b_i \leftarrow b_i + (\bar{d} - d)y_i x_{ij}, \ \forall i$.
     (d) If
$$|w_j + d| - |w_j| + C \sum_{i \in I(\boldsymbol{w} + d\boldsymbol{e}_j)} b_i^2 - L_{j,0} \le \sigma\Delta,$$
     then
     BREAK.
     Else
$$\bar{d} \leftarrow d; \ d \leftarrow \beta d; \ \Delta \leftarrow \beta\Delta.$$
  4. Update $w_j \leftarrow w_j + d$.

**Algorithm 5** A coordinate descent algorithm for L1-regularized logistic regression

- Choose $\beta = 0.5$ and $\sigma = 0.01$. Given initial $\boldsymbol{w} \in R^n$.
- Calculate
$$b_i \leftarrow e^{\boldsymbol{w}^T \boldsymbol{x}_i},\ i = 1, \ldots, l.$$
- While $\boldsymbol{w}$ is not optimal

  For $j = 1, 2, \ldots, n$

  1. Find the Newton direction by solving
$$\min_z \quad |w_j + z| + L'_j(0; \boldsymbol{w})z + \frac{1}{2}L''_j(0; \boldsymbol{w})z^2.$$
  The solution is
$$d = \begin{cases} -\frac{L'_j(0;\boldsymbol{w})+1}{L''_j(0;\boldsymbol{w})} & \text{if } L'_j(0; \boldsymbol{w}) + 1 \le L''_j(0; \boldsymbol{w})w_j, \\ -\frac{L'_j(0;\boldsymbol{w})-1}{L''_j(0;\boldsymbol{w})} & \text{if } L'_j(0; \boldsymbol{w}) - 1 \ge L''_j(0; \boldsymbol{w})w_j, \\ -w_j & \text{otherwise.} \end{cases}$$
  2. $\Delta \leftarrow L'_j(0; \boldsymbol{w})d + |w_j + d| - |w_j|$.
  3. While

     (a) If $\min_{i,j} x_{ij} \ge 0$ and
$$\min\big((55) - d \sum_{i:y_i=1} Cx_{ij} + |w_j + d| - |w_j|,\ (53) + d \sum_{i:y_i=-1} Cx_{ij} + |w_j + d| - |w_j|\big) \le \sigma\Delta,$$
     then
$$b_i \leftarrow b_i \times e^{dx_{ij}},\ \forall i \text{ and BREAK.}$$

     (b) $\bar{b}_i \leftarrow b_i \times e^{dx_{ij}},\ \forall i$.

     (c) If
$$\sum_{i:x_{ij}\neq 0} C \log\left(\frac{\bar{b}_i + 1}{\bar{b}_i + e^{dx_{ij}}}\right) + d \sum_{i:y_i=-1} Cx_{ij} + |w_j + d| - |w_j| \le \sigma\Delta,$$
     then
$$b_i \leftarrow \bar{b}_i,\ \forall i \text{ and BREAK.}$$
     Else
$$d \leftarrow \beta d;\ \Delta \leftarrow \beta\Delta.$$
  4. Update $w_j \leftarrow w_j + d$.

# A Practical Guide to LIBLINEAR

## Appendix N.

In this section, we present a practical guide for LIBLINEAR users. For instructions of using LIBLINEAR and additional information, see the `README` file included in the package and the LIBLINEAR FAQ, respectively.

### N.1 When to Use Linear (e.g., LIBLINEAR) Rather Then Nonlinear (e.g., LIBSVM)?

Please see the discussion in Appendix C of Hsu et al. (2003).

### N.2 Data Preparation (In Particular, Document Data)

LIBLINEAR can be applied to general problems, but it is particularly useful for document data. We discuss how to transform documents to the input format of LIBLINEAR.

A data instance of a classification problem is often represented in the following form.

label feature 1, feature 2, ..., feature $n$

The most popular method to represent a document is the "bag-of-words" model (Harris, 1954), which considers each word as a feature. Assume a document contains the following two sentences

This is a simple sentence.
This is another sentence.

We can transfer this document to the following Euclidean vector:

| a | an | $\cdots$ | another | $\cdots$ | is | $\cdots$ | this | $\cdots$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | $\cdots$ | 1 | $\cdots$ | 2 | $\cdots$ | 2 | $\cdots$ |

The feature "is" has a value 2 because "is" appears twice in the document. This type of data often has the following two properties.

1. The number of features is large.
2. Each instance is sparse because most feature values are zero.

Let us consider a 20-class data set news20 at libsvmtools.[6] The first instance is

```
1 197:2 321:3 399:1 561:1 575:1 587:1 917:1 1563:1 1628:3 1893:1 3246:1 4754:2
6053:1 6820:1 6959:1 7571:1 7854:2 8407:1 11271:1 12886:1 13580:1 13588:1
13601:2 13916:1 14413:1 14641:1 15950:1 20506:1 20507:1
```

Each (index:value) pair indicates a word's "term frequency" (TF). In practice, additional steps such as removing stop words or stemming (i.e., using only root words) may be applied, although we do not discuss details here.

Existing methods to generate document feature vectors include

6. `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#news20`

1. TF: term frequency.
2. TF-IDF (term frequency, inverse document frequency); see Salton and Yang (1973) and any information retrieval book.
3. binary: each feature indicates if a word appears in the document or not.

For example, the binary-feature vector of the news20 instance is

```
1 197:1 321:3 399:1 561:1 575:1 587:1 917:1 1563:1 1628:3 1893:1 3246:1 4754:1
6053:1 6820:1 6959:1 7571:1 7854:1 8407:1 11271:1 12886:1 13580:1 13588:1
13601:1 13916:1 14413:1 14641:1 15950:1 20506:1 20507:1
```

Our past experiences indicate that binary and TF-IDF are generally useful if normalization has been properly applied (see the next section).

### N.3 Normalization

In classification, large values in data may cause the following problems.

1. Features in larger numeric ranges may dominate those in smaller ranges.
2. Optimization methods for training may take longer time.

The typical remedy is to scale data feature-wisely. This is suggested in the popular SVM guide by Hsu et al. (2003).[7] However, for document data, often a simple instance-wise normalization is enough. Each instance becomes a unit vector; see the following normalized news20 instance.

```
1 197:0.185695 321:0.185695 399:0.185695 561:0.185695 575:0.185695 587:0.185695
917:0.185695 1563:0.185695 1628:0.185695 1893:0.185695 3246:0.185695 4754:0.185695
6053:0.185695 6820:0.185695 6959:0.185695 7571:0.185695 7854:0.185695 8407:0.185695
11271:0.185695 12886:0.185695 13580:0.185695 13588:0.185695 13601:0.185695
13916:0.185695 14413:0.185695 14641:0.185695 15950:0.185695 20506:0.185695 20507:0.185695
```

There are 29 non-zero features, so each one has the value $1/\sqrt{29}$. We run LIBLINEAR to find five-fold cross-validation (CV) accuracy on the news20 data.[8] For the original data using TF features, we have

```
$ time ./train -v 5 news20
Cross Validation Accuracy = 80.433%
23.709s
```

During the computation, the following warning message appears many times to indicate certain difficulties.

```
WARNING: reaching max number of iterations
```

7. For sparse data, scaling each feature to $[0, 1]$ is more suitable than $[-1, 1]$ because the latter makes the problem has many non-zero elements.
8. Experiments in this guide were run on a 64-bit machine with Intel Xeon E5-2620 2.00GHz CPU and LIBLINEAR 2.42 was used.

Table 1: Current solvers in LIBLINEAR for L2-regularized problems.

| | Dual-based solvers (coordinate descent methods) | Primal-based solvers (Newton-type methods) |
|---|---|---|
| Property | Extremely fast in some situations, but may be very slow in some others | Moderately fast in most situations |
| When to use it | 1. Large sparse data (e.g., documents) under suitable scaling and $C$ is not too large<br>2. Data with # instances $\ll$ # features | Others |

If data is transformed to binary and then normalized, we have

```
$ time ./train -v 5 news20.scale
Cross Validation Accuracy = 84.7129%
5.213s
```

Clearly, the running time for the normalized data is much shorter and no warning message appears.

Another example is in Lewis et al. (2004), which normalizes each log-transformed TF-IDF feature vector to have unit length.

### N.4 Selection of Solvers

Appendix A lists many LIBLINEAR's solvers. Users may wonder how to choose them. Fortunately, these solvers usually give similar performances. Some in fact generate the same model because they differ in only solving the primal or dual optimization problems. For example, `-s 1` solves dual L2-regularized L2-loss SVM, while `-s 2` solves primal.

```
$ ./train -s 1 -e 0.0001 news20.scale
(skipped)
Objective value = -576.922614
nSV = 3540
$./train -s 2 -e 0.0001 news20.scale
(skipped)
iter  8 f 5.769e+02 |g| 4.126e-02 CG   4 step_size 1.00e+00
iter  9 f 5.769e+02 |g| 6.076e-03 CG   7 step_size 1.00e+00
```

Each run trains 20 two-class models, so we only show objective values of the last one. Clearly, the dual objective value `-576.922614` coincides with the primal value `5.769e+02`. We use the option `-e` to impose a smaller stopping tolerance, so optimization problems are solved more accurately.

While LIBLINEAR's solvers give similar performances, their training time may be different. For current solvers for L2-regularized problems, a rough guideline is in Table 1. We recommend users

1. Try the default dual-based solver first.
2. If it is slow, check primal-based solvers.

For L1-regularized problems, the choice is limited because currently we only offer primal-based solvers.

To choose between using L1 and L2 regularization, we recommend trying L2 first unless users need a sparse model. In most cases, L1 regularization does not give higher accuracy but may be slightly slower in training; see a comparison in Section D of the supplementary materials of Yuan et al. (2010).

After version 2.42, we automatically switch from a dual-based solver to a primal one if the dual solver failed to converge after a pre-specified maximal number of iterations. For example, if `-s 1` is used to solve the dual problem of L2-regularized L2-loss SVM, we have

```
$ time ./train news20
(skipped)
WARNING: reaching max number of iterations
Switching to use -s 2

init f 3.516e+02 |g| 4.552e+02
iter  1 f 3.299e+02 |g| 9.844e+01 CG   4 step_size 1.00e+00
iter  2 f 3.284e+02 |g| 8.542e+01 CG   7 step_size 1.00e+00
iter  3 f 3.280e+02 |g| 2.699e+01 CG   2 step_size 1.00e+00
(skipped)
```

The coordinate descent method employed by `-s 1` converges slowly for this problem. LIBLINEAR then automatically switch to `-s 2`, which is a primal Newton method for the same optimization problem. Clearly we see that after switching the solver, the training procedure finishes in just a few iterations.

### N.5 Parameter Selection for Classification and Regression

For linear classification, the only parameter is $C$ in Eq. (1). We summarize some properties below.

1. Solvers in LIBLINEAR is *not very sensitive* to $C$. Once $C$ is larger than certain value, the obtained models have similar performances. A theoretical proof of this kind of results is in Theorem 3 of Keerthi and Lin (2003).
2. Using a larger $C$ value usually causes longer training time. Users should avoid trying such values.

We conduct the following experiments to illustrate these properties. To begin, we show in Figure 2 the relationship between $C$ and CV accuracy. Clearly, CV rates are similar after $C$ is large enough. For running time, we compare the results using $C = 1$ and $C = 100$.

```
$ time ./train -c 1 news20.scale
2.109s
$ time ./train -c 100 news20.scale
8.304s
```

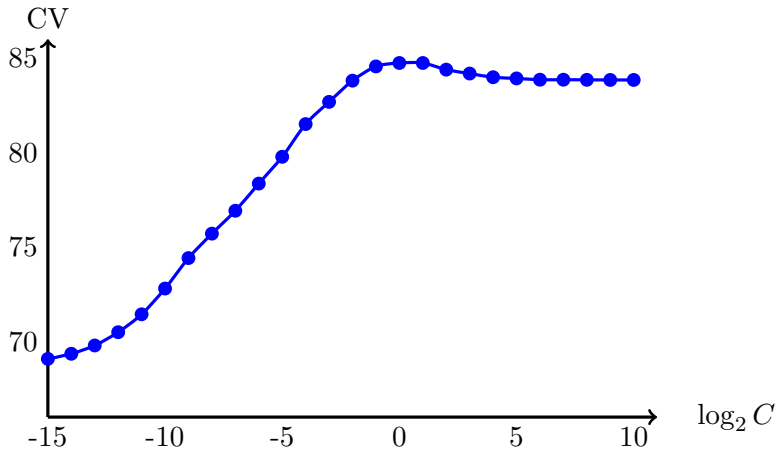



Figure 2: CV accuracy versus $\log_2 C$.

We can see that a larger $C$ leads to longer training time. Here a dual-based coordinate descent method is used. For primal-based solvers using Newton methods, the running time is usually less affected by the change of $C$.

```
$ time ./train -c 1 -s 2 news20.scale
5.013s
$ time ./train -c 100 -s 2 news20.scale
5.448s
```

While users can try a few $C$ values by themselves, LIBLINEAR (after version 2.0) provides an easy solution to find a suitable parameter. If primal-based classification solvers are used (`-s 0` or `-s 2`), the `-C` option efficiently conducts cross validation several times and finds the best parameter automatically.

```
$ ./train -C -s 2 news20.scale
Doing parameter search with 5-fold cross validation.
log2c= -15.00 rate=69.0806
log2c= -14.00 rate=69.3505
(skipped)
log2c=   9.00 rate=83.8406
log2c=  10.00 rate=83.8406
WARNING: maximum C reached.
Best C = 1  CV accuracy = 84.738%
```

Users do not need to specify the range of $C$ to explore because LIBLINEAR finds a reasonable range automatically. Note that users can still use the `-c` option to specify the smallest $C$ value of the search range. For example,

```
$ ./train -C -s 2 -c 0.5 -e 0.0001 news20.scale
Doing parameter search with 5-fold cross validation.
log2c=  -1.00 rate=84.5623
```

```
log2c=   0.00 rate=84.7254
(skipped)
log2c=   9.00 rate=83.6586
log2c=  10.00 rate=83.6398
WARNING: maximum C reached.
Best C = 1  CV accuracy = 84.7254%
```

This option is useful when users want to rerun the parameter selection procedure from a specified $C$ under a different setting, such as a stricter stopping tolerance `-e 0.0001` in the above example.

Note that after finding the best $C$, users must apply the same solver to train a model for future prediction. Switching from a primal-based solver to a corresponding dual-based solver (e.g., from `-s 2` to `-s 1`) is fine because they produce the same model.

Some solvers such as `-s 5` or `-s 6` are not covered by the `-C` option.[9] We can use a parameter selection tool `grid.py` in LIBSVM to find the best $C$ value. For example,

```
$ ./grid.py -log2c -14,14,1 -log2g null -svmtrain ./train -s 5 news20.scale
```

checks the CV rates of $C \in \{2^{-14}, 2^{-13}, \ldots, 2^{14}\}$. Note that `grid.py` should be used only if `-C` is not available for the desired solver. The `-C` option is much faster than `grid.py` on a single computer.

For regression, an additional parameter is $\epsilon$. After version 2.30, the `-C` option automatically finds the best $\epsilon$ as well. The usage is the same though the `-p` option can be used to specify the largest $\epsilon$ value of the search range. For example, we can use

```
$ ./train -C -s 11 -c 0.5 -p 2 -e 0.0001 news20.scale
```

so that the search range consists of $C \in [0.5, \infty)$ and $\epsilon \in [0, 2]$.

## References

Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. Coordinate descent method for large-scale L2-loss linear SVM. *Journal of Machine Learning Research*, 9:1369–1398, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/cdl2.pdf`.

Bo-Yu Chu, Chia-Hua Ho, Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. Warm start for parameter selection of linear classifiers. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2015. URL `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/warm-start/warm-start.pdf`.

Koby Crammer and Yoram Singer. On the learnability and design of output codes for multiclass problems. In *Computational Learning Theory*, pages 35–46, 2000.

Jerome H. Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

9. Note that we have explained that for some dual-based solvers you can use `-C` on their corresponding primal-based solvers for parameter selection.

Leonardo Galli and Chih-Jen Lin. Truncated Newton methods for linear classification. Technical report, National Taiwan University, 2020a. URL https://www.csie.ntu.edu.tw/~cjlin/papers/tncg/tncg.pdf.

Leonardo Galli and Chih-Jen Lin. Release note of liblinear 2.40. Technical report, National Taiwan University, 2020b. URL https://www.csie.ntu.edu.tw/~cjlin/papers/tncg/release240.pdf.

Zellig S. Harris. Distributional structure. *Word*, 10:146–162, 1954.

Chia-Hua Ho and Chih-Jen Lin. Large-scale linear support vector regression. *Journal of Machine Learning Research*, 13:3323–3348, 2012. URL http://www.csie.ntu.edu.tw/~cjlin/papers/linear-svr.pdf.

Chih-Yang Hsia, Ya Zhu, and Chih-Jen Lin. A study on trust region update rules in Newton methods for large-scale linear classification. In *Proceedings of the Asian Conference on Machine Learning (ACML)*, 2017. URL http://www.csie.ntu.edu.tw/~cjlin/papers/newtron/newtron.pdf.

Chih-Yang Hsia, Wei-Lin Chiang, and Chih-Jen Lin. Preconditioned conjugate gradient methods in truncated Newton frameworks for large-scale linear classification. In *Proceedings of the Asian Conference on Machine Learning (ACML)*, 2018. URL http://www.csie.ntu.edu.tw/~cjlin/papers/tron_pcg/precondition.pdf.

Jui-Yang Hsia and Chih-Jen Lin. Parameter selection for linear support vector regression. *IEEE Transactions on Neural Networks and Learning Systems*, 2020. URL https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/warm-start/svr-param.pdf. To appear.

Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the Twenty Fifth International Conference on Machine Learning (ICML)*, 2008. URL http://www.csie.ntu.edu.tw/~cjlin/papers/cddual.pdf.

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003. URL http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf.

Tzu-Kuo Huang, Ruby C. Weng, and Chih-Jen Lin. Generalized Bradley-Terry models and multi-class probability estimates. *Journal of Machine Learning Research*, 7:85–115, 2006. URL http://www.csie.ntu.edu.tw/~cjlin/papers/generalBT.pdf.

S. Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with Gaussian kernel. *Neural Computation*, 15(7):1667–1689, 2003.

S. Sathiya Keerthi, Sellamanickam Sundararajan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A sequential dual method for large scale multi-class linear SVMs. In *Proceedings of the Forteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 408–416, 2008. URL http://www.csie.ntu.edu.tw/~cjlin/papers/sdm_kdd.pdf.

Ching-Pei Lee and Chih-Jen Lin. A study on L2-loss (squared hinge-loss) multi-class SVM. *Neural Computation*, 25(5):1302–1323, 2013. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/l2mcsvm/l2mcsvm.pdf`.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5: 361–397, 2004.

Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. Trust region Newton method for large-scale logistic regression. *Journal of Machine Learning Research*, 9:627–650, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/logistic.pdf`.

Gerard Salton and Chung-Shu Yang. On the specification of term values in automatic indexing. *Journal of Documentation*, 29:351–372, 1973.

Paul Tseng and Sangwoon Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117:387–423, 2009.

Hsiang-Fu Yu, Fang-Lan Huang, and Chih-Jen Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1-2):41–75, October 2011. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/maxent_dual.pdf`.

Guo-Xun Yuan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *Journal of Machine Learning Research*, 11:3183–3234, 2010. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/l1.pdf`.

Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. An improved GLMNET for l1-regularized logistic regression. *Journal of Machine Learning Research*, 13:1999–2030, 2012. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/l1_glmnet/long-glmnet.pdf`.