LIBOL: A Library for Online Learning Algorithms

Version 0.2.0

Steven C.H. Hoi
Jialei Wang
Peilin Zhao
School of Computer Engineering
Nanyang Technological University
Singapore 639798
E-mail: chhoi@ntu.edu.sg

`http://LIBOL.stevenhoi.org/`

July 27, 2013

Abstract

LIBOL is an open-source library for large-scale online classification, which consists of a large family of efficient and scalable state-of-the-art online learning algorithms for large-scale online classification tasks. We have offered easy-to-use command-line tools and examples for users and developers. We also have made comprehensive documents available for both beginners and advanced users. LIBOL is not only a machine learning tool, but also a comprehensive experimental platform for conducting online learning research.

Contents

1 Introduction

Online learning represents an important family of efficient and scalable machine learning algorithms for large-scale applications. In general, online learning algorithms are fast, simple, and often make few statistical assumptions, making them applicable to a wide range of applications. Online learning has been actively studied in several communities, including machine learning, statistics, and artificial intelligence. Over the past years, a variety of online learning algorithms have been proposed, but so far there is very few comprehensive library which includes most of the state-of-the-art algorithms for researchers to make easy side-by-side comparisons and for developers to explore their various applications.

In this work, we develop LIBOL as an easy-to-use online learning tool that consists a large family of existing and recent state-of-the-art online learning algorithms for large-scale online classification tasks. In contrast to many existing software for large-scale data classification, LIBOL enjoys significant advantages for massive-scale classification in the era of big data nowadays, especially in efficiency, scalability, parallelization, and adaptability. The software is available at `http://libol.stevenhoi.org/`.

1.1 Suitable Tasks

Currently the implemented toolbox is suitable for online binary classification and multi-calss classification tasks, which are discussed with more details as follows.

1.1.1 Binary Classification

Online binary classification operates on a sequence of data examples with time stamps. At each step t, the learner receives an incoming example $\mathbf{x}_t \in \mathcal{X}$ in a d-dimensional vector space, i.e., $\mathcal{X} = \mathbb{R}^d$. It first attempts to predict the class label of the incoming instance,

$$\hat{y}_t = \mathsf{sgn}(f(\mathbf{x}_t; \mathbf{w}_t)) = \mathsf{sgn}(\mathbf{w}_t \cdot \mathbf{x}_t) \in \mathcal{Y}$$

and $\mathcal{Y} = \{-1, +1\}$ for binary classification tasks. After making the prediction, the true label $y_t \in \mathcal{Y}$ is revealed, and the learners then computes the loss $\ell(y_t, \hat{y}_t)$ based on some criterion to measure the difference between the learner's prediction and the revealed true label y_t. Based on the result of the loss, the learner finally decides when and how to update the classification model at the end of each learning step. The following algorithmic framework gives an overview of most online learning algorithms [1] for linear classification, where $\Delta(\mathbf{w}_t; (\mathbf{x}_t, y_t))$ denotes the update of the classification models. Different online learning algorithms in general are distinguished in terms of different definitions and designs of the loss function $\ell(\cdot)$ and their various updating functions $\Delta(\cdot)$.

In particular, this software consists of 16 different online algorithms and their variants for binary classification, and 13 online algorithms and variants for multiclass classification. In general, they can be grouped into two major categories: (i) first-order

[1] Except that SOP in [1] follows a slightly different procedure.

Algorithm 1: LIBOL: Online Learning Framework for Linear Classification.

```
1  Initialize: $\mathbf{w}_1 = 0$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\hat{y}_t = \mathsf{sgn}(f(\mathbf{x}_t; \mathbf{w}_t))$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $\ell(\mathbf{w}_t; (\mathbf{x}_t, y_t))$;
7      if $\ell(\mathbf{w}_t; (\mathbf{x}_t, y_t)) > 0$ then
8          The learner updates the classification model:
9          $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \Delta(\mathbf{w}_t; (\mathbf{x}_t, y_t))$;
10     end
11 end
```

learning [13, 2], and (ii) second-order learning [4, 14, 15]. Examples of online learning algorithms in the first-order learning category include the following list of classical and popular algorithms:

- Perceptron: the classical online learning algorithm [13];
- ALMA: Approximate Maximal Margin Classification Algorithm [10];
- ROMMA: the Relaxed Online Maxiumu Margin algorithms [11];
- OGD: the Online Gradient Descent (OGD) algorithms [18];
- PA: Passive Aggressive (PA) algorithms [2], one of state-of-the-art first-order online learning algorithms;

In recent years, to improve the first-order learning methods, the second-order online learning algorithms typically assume the weight vector follows a Gaussian distribution $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ with mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$. The model parameters, including both the mean vector and the covariance matrix are updated in the online learning process. Example of the second-order online learning algorithms include the following:

- SOP: the Second-Order Perceptron (SOP) algorithm [1];
- CW: the Confidence-Weighted (CW) learning algorithm [4];
- IELLIP: online learning algorithms by improved ellipsoid method [15];
- AROW: the Adaptive Regularization of Weight Vectors [5];
- NAROW: New variant of Adaptive Regularization [12];
- NHERD: the Normal Herding method via Gaussian Herding [7]
- SCW: the recently proposed Soft Confidence Weighted algorithms [14].

For the details of each of the above algorithms, please refer to the detailed descriptions in Section 2.

1.1.2 Multiclass Classification

Similar to online binary classification, online *multiclass* learning is performed over a sequence of training examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$. Unlike binary classification where $y_t \in \{-1, +1\}$, in multiclass learning, each class assignment $y_t \in \mathcal{Y} = \{1, \ldots, k\}$, making it a more challenging problem. We use $\hat{y}_t$ to represent the class label predicted by the online learning algorithm. Online multiclass classification algorithms [8, 9] learn multiple hypotheses/classifiers, one classifier for each class in $\mathcal{Y}$, leading to a total of k classifiers that are trained for the classification task. The predicted label is the one, which is associated with the largest prediction value, i.e.

$$\widehat{y}_t = \arg \max_{i \in \{1,\ldots,k\}} \mathbf{w}_{t,i} \cdot \mathbf{x}_t.$$

After the prediction, the true label $y_t \in \mathcal{Y}$ will be disclosed, the learner then compute the loss based on some criterion to measure the difference between the prediction and the true label. Based on the results of the loss, the learner finally decides when and how to update the k classifiers at the end of each learning step.

In particular, this software package consists of 13 different online multiclass classification algorithms and their variants. In general, they can be grouped into two major categories: (i) first-order learning [8, 2], and (ii) second-order learning [3, 6].

Specifically, the first-order learning algorithms only keep updating k classification functions (for the k different labels), which only utilize the first-order information of the received instances. The main first-order algorithms implemented in this toolbox include:

- Perceptron: the classical multiclass online learning algorithm [8];
- ROMMA: the Multiclass Relaxed Online Maxiumu Margin algorithms [11];
- OGD: the Multiclass Online Gradient Descent (OGD) algorithms [18];
- PA: Multiclass Passive Aggressive (PA) algorithms [2];

Unlike the first-order learning algorithms, the second-order online learning aim to better exploit the underlying structures between features. Specifically, the second-order online learning algorithms typically assume the weight vector follows Gaussian distributions $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ with mean vector $\boldsymbol{\mu} \in \mathbb{R}^{kd}$ and covariance matrix $\Sigma \in \mathbb{R}^{kd \times kd}$. The main second-order algorithms implemented in this toolbox include:

- CW: the Multiclass Confidence-Weighted (CW) learning algorithm [3];
- AROW: the Multiclass Adaptive Regularization of Weight Vectors [6];
- SCW: the recently proposed Soft Confidence Weighted algorithms [**?**].

For the details of all the algorithms, please refer to the section 2.

1.2 Main Features

- **Comprehensiveness:** A large family of existing online binary and multiclass classification algorithms have been implemented in this software package;
- **Extendibility:** One can easily implement a new algorithm by only implementing the key updating module;
- **Usability:** It is easy to use the main functions to evaluate one algorithm by comparing with all the algorithms;

1.3 Summary of Main Algorithms

Table 1 gives a summary of the family of implemented algorithms in this software package. For simplicity, we exclude some incremental variants of the algorithms in this list.

Table 1: Summary of the Implemented Algorithms.

Problem Type	Methodology	Algorithm	Description	Section
Binary Classification	First-Order	Perceptron	The perceptron algorithm [13]	2.1.1
		ALMA	Approximate Large Margin Algorithm [10]	2.1.2
		ROMMA	Relaxed Online Maxiumu Margin Algorithms [11]	2.1.3
		OGD	Online Gradient Descent [18]	2.1.4
		PA	Passive Aggressive (PA) algorithms [2]	2.1.5
		SOP	Second-Order Perceptron [1]	2.2.1
	Second-Order	CW	Confidence-Weighted (CW) learning [4]	2.2.2
		IELLIP	Improved Ellipsoid method [15]	2.2.3
		AROW	Adaptive Regularization of Weight Vectors [5]	2.2.4
		NAROW	New variant of Adaptive Regularization [12]	2.2.5
		NHERD	Normal Herding method via Gaussian Herding [7]	2.2.6
		SCW	Soft Confidence Weighted algorithms [14]	2.2.7
Multiclass Classification	First-Order	Perceptron	The perceptron algorithm [13]	3.1.1
		ROMMA	Approximate Large Margin Algorithm [10]	3.1.2
		OGD	Online Gradient Descent [18]	3.1.3
		PA	Passive Aggressive (PA) algorithms [2]	3.1.4
	Second-Order	CW	Confidence-Weighted (CW) learning [4]	3.2.1
		AROW	Adaptive Regularization of Weight Vectors [5]	3.2.2
		SCW	Soft Confidence Weighted algorithms [14]	3.2.3

1.4 How to Use the LIBOL Package

The current version of LIBOL package includes a MATLAB library, a C library and command-line tools for the learning task. The data formats used by this software package are compatible to some of the very popular machine learning and data mining packages, such as LIBSVM, SVM-light, and WEKA, etc.

To evaluate an algorithm on a dataset with a specific format, one can call the function:

$$\$\ \texttt{demo}(TaskType, Algorithm, Dataset, DataFormat)$$

where the details of the input parameters are listed as in the Table 2.

To easily compare all the online learning algorithms for one problem type on one dataset, you can call the functions as

$$\$\ \texttt{run\_experiment}(TaskType, Dataset, DataFormat)$$

where the details of the input parameters are listed as in the Table 2.

Table 2: The input parameters for function "demo.m".

Parameter	Description
TaskType	The type of the online learning problem. Two optional values: 'bc' and 'mc'. 'bc' denotes binary classification. 'mc' denotes multiclass classification.
Algorithm	The name of the evaluated algorithm, which is in the folder - algorithms. For example, 'Perceptron' and 'Perceptron_c' for the 'bc' problem. 'Perceptron' and 'Perceptron_c' are Matlab and C versions, respectively. For example, 'PerceptronM' for the 'mc' problem.
Dataset	The name of the dataset for evaluation For example: svmguide3. For example: 'ionosphere.arff'.
DataFormat	The format of the dataset. By default, use 'mat' version 'libsvm' and 'arff' denote the format for LIBSVM and WEKA

1.4.1 Folders

The Matlab and C versions of the algorithms listed in subsection 1.1 are located at the folders "\LIBOL\algorithms" and "\LIBOL\algorithms_mex", respectively. The main functions "demo" and "run_experiment" are located at the folder "\LIBOL". To use these functions, you can use the command "addpath(genpath(['root\LIBOL']));", where "root" is the path to the folder "LIBOL".

All the datasets to be evaluated on should be placed into the folder "\LIBOL\data". The dataset can be in formats of mat, LIBSVM and WEKA versions.

1.4.2 Examples

To illustrate the online learning procedure, we take two data sets from the LIBSVM website, including one small data set "svmguide3" with 1243 instances and one large data set "ijcnn1" with 141,691 instances. In the following example, we use the default "Perceptron" algorithm to demo the usage of LIBOL for a binary classification ('bc') task:

```
$ demo('bc', 'Perceptron', 'svmguide3')
```

The results output by the above command are summarized as follows:

```
Algorithm:   mistake rate          nb of updates       cpu time (seconds)
Perceptron   0.3296 +/- 0.0103     409.75 +/- 12.80    0.0458 +/- 0.0006
```

To ease researchers to run a full set of experiments for side-by-side comparisons of different algorithms, we offer a very easy-to-use example program as follows:

```
$ run_experiment('bc', 'svmguide3')
```

The above command will run side-by-side comparison of varied online learning algorithms on the given data set fully auotmatically, including all the parameter settings and selection. The full set of experimental results will be generated by the library automatially, as shown in Table 1 and Figure 1.

Table 3: Comparison of a variety of online learning algorithms on two data sets.

Dataset:	svmguide3 (#samples=1243,#dimensions=36)			ijcnn1 (#samples=141,691,#dimensions=22)		
Algorithm	mistake	# updates	time (s)	mistake	# updates	time (s)
Perceptron	0.330 ± 0.010	409.8 ± 12.8	0.045 ± 0.000	0.106 ± 0.001	15052.9 ± 71.2	5.452 ± 0.128
ROMMA	0.338 ± 0.013	419.9 ± 16.4	0.049 ± 0.000	0.101 ± 0.001	14291.8 ± 72.2	5.720 ± 0.160
aROMMA	0.333 ± 0.014	516.3 ± 24.1	0.053 ± 0.001	0.101 ± 0.001	14806.6 ± 98.5	5.542 ± 0.194
ALMA	0.237 ± 0.007	586.4 ± 10.0	0.059 ± 0.001	0.072 ± 0.000	21753.0 ± 58.7	6.671 ± 0.165
OGD	0.238 ± 0.003	637.5 ± 3.3	0.060 ± 0.001	0.095 ± 0.000	27449.6 ± 45.1	6.692 ± 0.174
PA1	0.236 ± 0.002	763.9 ± 9.9	0.063 ± 0.001	0.077 ± 0.000	28399.0 ± 83.5	6.766 ± 0.230
PA2	0.254 ± 0.005	1134.1 ± 13.3	0.071 ± 0.001	0.081 ± 0.000	61085.9 ± 154.8	18.961 ± 0.895
SOP	0.294 ± 0.009	365.9 ± 11.7	0.104 ± 0.001	0.102 ± 0.001	14487.8 ± 86.0	11.890 ± 0.348
IELLIP	0.317 ± 0.009	393.4 ± 11.4	0.074 ± 0.002	0.119 ± 0.001	16812.3 ± 135.2	7.978 ± 0.249
CW	0.295 ± 0.008	699.9 ± 13.3	0.094 ± 0.001	0.093 ± 0.001	30678.0 ± 146.9	11.181 ± 0.394
NHERD	0.224 ± 0.013	1152.1 ± 21.9	0.102 ± 0.001	0.084 ± 0.001	84878.5 ± 4007.8	36.555 ± 3.098
AROW	0.225 ± 0.005	1219.5 ± 6.5	0.127 ± 0.001	0.082 ± 0.000	73563.1 ± 1318.8	29.979 ± 1.460
NAROW	0.267 ± 0.032	1181.5 ± 39.1	0.133 ± 0.003	0.097 ± 0.013	103203.7 ± 8566.6	53.171 ± 7.636
SCW	0.214 ± 0.006	583.9 ± 14.9	0.084 ± 0.001	0.058 ± 0.002	10829.3 ± 524.0	7.508 ± 0.262
SCW2	0.215 ± 0.008	784.0 ± 70.5	0.093 ± 0.003	0.070 ± 0.001	30965.5 ± 2433.9	9.518 ± 0.447

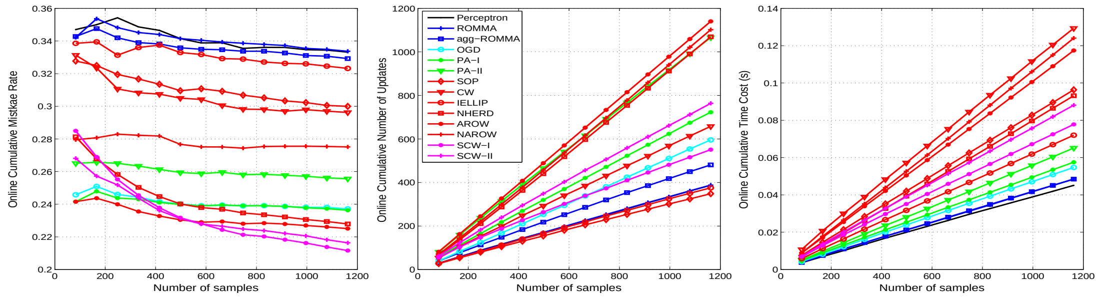

Figure 1: Comparison of a variety of online learning algorithms on dataset "svmguide3".

This library provides a fairly easy-to-use testbed to facilitate online learning researchers to develop their new algorithms and conduct side-by-side comparisons with the state-of-the-art algorithms with the minimal efforts.

1.5 Documentation

The LIBOL package comes with comprehensive documentation. The README file describes the setup and usage. Users can read the "Quick Start" section to begin shortly. All the functions and related data structures are explained in detail. If the README file does not give the information users want, they can check the online FAQ. In addition to software documentation, theoretical properties of the algorithms and comparisons can be found in [14]. The authors are also willing to answer any further questions.

1.6 Design

The design principle is to keep the package simple, easy to read and extend. All codes follow the MATLAB standard and need no external libraries, except for the support of popular data formats, such as LIBSVM and WEKA datasets for which existing libraries are included. All the online learning algorithms can be called via the uniform "`ol_train()`" function by setting proper options. In general, LIBOL is written in a modular way, in which one can easily develop a new algorithm and make side-by-side comparisons with the existing ones in the package. Thus, LIBOL is not only a machine learning tool, but also an experimental platform for online learning research.

2 Online Learning Algorithms for Binary Classification

In a regular binary classification task, the goal of online learning for classification is to minimize the cumulative mistake number over the entire sequence of data examples. There are some other atypical settings where other learning objectives are preferred. In literature, many linear algorithms have been proposed for online classification. They can be generally grouped into two major categories: (i) first-order online learning algorithms, and (ii) second-order online learning algorithms; Below we review the basics of these algorithms in detail.

2.1 First-Order Algorithms

In this section, we provide the details of the first-order learning algorithms and the corresponding functions(located at /algorithms/Binary-Classification). The first-order learning algorithms only keep updating one classification function, which only utilizes the first-order information of the examples.

Specifically, the struct "model" in the codes contains a field "**w**", which is the linear classifier updated round by round. In addition, "model" also contains various parameters for different algorithms, which will be explained in every algorithm.

2.1.1 Perceptron

The Perceptron [13] algorithm is the earliest and simplest approach for online learning.

Algorithm 2: Perceptron.

```
1  Initialize: $\mathbf{w}_1 = 0$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\widehat{y}_t = \text{sign}(\mathbf{w}^\top \mathbf{x}_t)$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}(\widehat{y}_t \neq y_t)$;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $\mathbf{w} = \mathbf{w} + l_t y_t \mathbf{x}_t$;
10     end
11 end
```

Perceptron does not contain any parameters.

2.1.2 Approximate Large Margin Algorithm (ALMA)

The Approximate Large Margin Algorithm(ALMA) [10] would try to learn an approximate large margin classifier.

Algorithm 3: ALMA: Approximate Large Margin Algorithm .

```
1  Initialize: $\mathbf{w}_1 = 0$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\widehat{y}_t = \mathrm{sign}(\mathbf{w}^\top \mathbf{x}_t)$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}(\max(0, \frac{1-\alpha}{\alpha\sqrt{k}} - \frac{y_t \mathbf{w}_t \cdot \mathbf{x}_t}{\|\mathbf{x}_t\|}) > 0)$;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $\mathbf{w} = (\mathbf{w} + l_t \sqrt{2/k} y_t \frac{\mathbf{x}_t}{\|\mathbf{x}_t\|}) / \max(1, \|\mathbf{w} + l_t \sqrt{2/k} y_t \frac{\mathbf{x}_t}{\|\mathbf{x}_t\|}\|)$
10         $k = k + l_t$;
11     end
12 end
```

The implemented algorithm is ALMA$_2(\alpha)$, so that "model" contains the parameter:

- $\alpha \in (0, 1]$: the final margin is $(1 - \alpha)\gamma_*$, where γ_* is the best margin

2.1.3 Relaxed Online Maximum Margin Algorithm (ROMMA)

The Relaxed Online Maximum Margin Algorithm(ROMMA) [11] and its aggressive version agg-ROMMA.

Algorithm 4: ROMMA: Relaxed Online Maximum Margin Algorithm.

```
1  Initialize: $\mathbf{w}_1 = 0$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\widehat{y}_t = \mathrm{sign}(\mathbf{w}^\top \mathbf{x}_t)$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss:
           $$l_t = \begin{cases} \mathbb{I}(\widehat{y}_t \neq y_t) & \text{ROMMA ,} \\ \mathbb{I}(\max(0, 1 - y_t(\mathbf{w}_t \cdot \mathbf{x}_t)) > 0) & \text{agg-ROMMA .} \end{cases}$$
       if $l_t > 0$ then
7          The learner updates the classification model:
8          $\mathbf{w}_{t+1} = (\frac{\|\mathbf{x}_t\|^2 \|\mathbf{w}_t\|^2 - y_t \mathbf{w}_t \cdot \mathbf{x}_t}{\|\mathbf{x}_t\|^2 \|\mathbf{w}_t\|^2 - (\mathbf{w}_t \cdot \mathbf{x}_t)^2}) \mathbf{w}_t + (\frac{\|\mathbf{w}_t\|^2 (y_t - \mathbf{w}_t \cdot \mathbf{x}_t)}{\|\mathbf{x}_t\|^2 \|\mathbf{w}_t\|^2 - (\mathbf{w}_t \cdot \mathbf{x}_t)^2}) \mathbf{x}_t$;
9      end
10 end
```

The ROMMA and agg-ROMMA algorithm do not contains parameters.

2.1.4 Online Gradient Descent (OGD) Algorithm

The online gradient descent algorithm [18] applies the gradient descent updating approach together with the hinge loss.

Algorithm 5: OGD: Online Gradient Descent.

```
1  Initialize: $\mathbf{w}_1 = 0$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\widehat{y}_t = \mathrm{sign}(\mathbf{w}^\top \mathbf{x}_t)$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}(\max(0, 1 - y_t(\mathbf{w}_t \cdot \mathbf{x}_t)) > 0)$;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $\mathbf{w}_{t+1} = \mathbf{w}_t + l_t \sqrt{1/t} y_t \mathbf{x}_t$;
10     end
11 end
```

In our implementation, the algorithm does not contains any parameters.

2.1.5 Passive-Aggressive learning (PA)

The family of online Passive-Aggressive (PA) learning [2] is formulated to trade off the objective of minimizing the distance between the learnt classifier and the previous classifier, and the objective of minimizing the loss of the learnt classier suffered on the current instance.

Formally the PA algorithm is formulated as the following online optimization problem

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|^2 \ , \quad \text{s.t. } \ell(\mathbf{w}; (\mathbf{x}_t, y_t)) = \max(0, 1 - y_t(\mathbf{w} \cdot \mathbf{x}_t)) = 0$$

Furthermore, PA is extended to PA-I and PA-II algorithms, which can better handle the non-separable and noisy case. PA-I is formulated as

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|^2 + C\ell(\mathbf{w}; (\mathbf{x}_t, y_t))$$

where $\ell(\mathbf{w}; (\mathbf{x}_t, y_t)) = \max(0, 1 - y_t(\mathbf{w} \cdot \mathbf{x}_t))$ and $C > 0$, which is used to trade of the passiveness and aggressiveness.

Finally, PA-II is formulated as

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w} - \mathbf{w}_t\|^2 + C(\ell(\mathbf{w}; (\mathbf{x}_t, y_t)))^2$$

where $\ell(\mathbf{w}; (\mathbf{x}_t, y_t)) = \max(0, 1 - y_t(\mathbf{w} \cdot \mathbf{x}_t))$ and $C > 0$, which is used to trade of the passiveness and aggressiveness.

The struct "model" contains the following parameter

- $C > 0$, which trades off the passiveness and aggressiveness

Algorithm 6: PA: Passive Aggressive algorithms.

```
1  Initialize: $\mathbf{w}_1 = 0$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\widehat{y}_t = \text{sign}(\mathbf{w}^\top \mathbf{x}_t)$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}(\max(0, 1 - y_t(\mathbf{w}_t \cdot \mathbf{x}_t)) > 0)$;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $\mathbf{w}_{t+1} = \begin{cases} \mathbf{w}_t + \frac{\ell(\mathbf{w}_t;(\mathbf{x}_t,y_t))}{\|\mathbf{x}_t\|^2} y_t \mathbf{x}_t & \text{PA} , \\ \mathbf{w}_t + \min\{C, \frac{\ell(\mathbf{w}_t;(\mathbf{x}_t,y_t))}{\|\mathbf{x}_t\|^2}\} y_t \mathbf{x}_t & \text{PA-I} , \\ \mathbf{w}_t + \frac{\ell(\mathbf{w}_t;(\mathbf{x}_t,y_t))}{\|\mathbf{x}_t\|^2 + \frac{1}{2C}} y_t \mathbf{x}_t & \text{PA-II.} \end{cases}$
10     end
11 end
```

2.2 Second-Order Algorithms

To better exploring the underlying structure between features, the second-order online learning algorithms typically assume the weight vector follows a Gaussian distribution $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ with mean vector $\boldsymbol{\mu} \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$.

Specifically, the struct "model" in the codes contains two fields "$\boldsymbol{\mu}$" and "Σ", which are the mean vector and covariance matrix for the learnt Gaussian distribution. In addition, "model" also contains various parameters for different algorithms, which will be explained in every algorithm.

2.2.1 Second-Order Perceptron (SOP)

The Second-Order Perceptron(SOP) [1] could be considered as the second-order counterpart of Perceptron by maintain a covariance matrix Σ_t.

The struct "model" contains the following parameter:

- $a > 0$, which is used to initialize the $\Sigma_1 = a\mathbf{I}$, where $\mathbf{I}$ is identity matrix.

2.2.2 Confidence-weighted learning (CW)

In confidence-weighted (CW) learning [4], the weight distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ is updated by minimizing the Kullback-Leibler divergence between the new weight distribution and the old one while ensuring that the probability of correct classfication is greater than a

Algorithm 7: SOP: Second-Order Perceptron.

```
1  Initialize: $\mathbf{w}_1 = 0$, $\Sigma_1 = a\mathbf{I}$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\widehat{y}_t = \mathrm{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$, where
       $\mathbf{w}_t = (\Sigma_t + \mathbf{x}_t\mathbf{x}_t^\top)^{-1}\boldsymbol{\mu}_t$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}(\widehat{y}_t \neq y_t)$;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + l_t y_t \mathbf{x}_t, \quad \Sigma_{t+1} = \Sigma_t + l_t \mathbf{x}_t \mathbf{x}_t^\top$;
10     end
11 end
```

threshold as follows:

$$(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) = \arg\min_{\boldsymbol{\mu},\Sigma} D_{KL}(\mathcal{N}(\boldsymbol{\mu}, \Sigma), \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t))$$
$$s.t. \quad Pr_{\mathbf{w}\sim\mathcal{N}(\boldsymbol{\mu},\Sigma)}[y_t(\mathbf{w}\cdot\mathbf{x}_t) \geq 0] \geq \eta$$

Algorithm 8: CW: Confidence Weighted algorithm.

```
1  Initialize: $\mathbf{w}_1 = 0$, $\Sigma_1 = a\mathbf{I}$, ($\Phi$ is the cumulative function of the normal
   distribution), $\psi = 1 + \frac{\phi^2}{2}$, and $\zeta = 1 + \phi^2$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\widehat{y}_t = \mathrm{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$, where $\mathbf{w}_t = \boldsymbol{\mu}_t$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner computes the updating coefficients:
7      $u_t = \frac{1}{4}(-\alpha_t\upsilon_t\phi + \sqrt{\alpha_t{}^2\upsilon_t{}^2\phi^2 + 4\upsilon_t})^2$, $\upsilon_t = \mathbf{x}_t{}^T\Sigma_t\mathbf{x}_t$, $m_t = y_t(\boldsymbol{\mu}_t \cdot \mathbf{x}_t)$
8      $\alpha_t = \max\left\{0, \frac{1}{\upsilon_t\zeta}(-m_t\psi + \sqrt{m_t{}^2\frac{\phi^4}{4} + \upsilon_t\phi^2\zeta})\right\}, \quad \beta_t = \frac{\alpha_t\phi}{\sqrt{u_t} + \upsilon_t\alpha_t\phi}$;
9      The learner calculates the suffered loss: $l_t = \mathbb{I}(\alpha_t > 0)$;
10     if $l_t > 0$ then
11         The learner updates the classification model:
12         $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \alpha_t y_t \Sigma_t \mathbf{x}_t$ , $\quad \Sigma_{t+1} = \Sigma_t - \beta_t\Sigma_t\mathbf{x}_t\mathbf{x}_t{}^\top\Sigma_t$;
13     end
14 end
```

The struct "model" contains the following parameters:

- $a > 0$, which is used to initialize the $\Sigma_1 = a\mathbf{I}$, where $\mathbf{I}$ is identity matrix;

- $\eta \in (0.5, 1]$, which is the probability required for the updated distribution on the current instance

2.2.3 Improved Ellipsoid Method (IELLIP)

The improved ellipsoid method [15] is an improved version of the classical ellipsoid method for online learning, which is modified so that it is able to address the inseparable case.

Algorithm 9: IELLIP: Improved Ellipsoid Method.

```
1  Initialize: $\mathbf{w}_1 = 0$, $\Sigma_1 = a\mathbf{I}$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\widehat{y}_t = \text{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$, where $\mathbf{w}_t = \boldsymbol{\mu}_t$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}(\widehat{y}_t \neq y_t)$;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \alpha_t \Sigma_t \mathbf{g}_t$ ,   $\Sigma_{t+1} = \frac{1}{1-c_t}(\Sigma_t - c_t \Sigma_t \mathbf{g}_t \mathbf{g}_t{}^\top \Sigma_t)$
10         $\alpha_t = \frac{\alpha\gamma - y_t \boldsymbol{\mu}_t^\top \mathbf{x}_t}{\sqrt{\mathbf{x}_t^\top \Sigma_t \mathbf{x}_t}}$,   $\mathbf{g}_t = \frac{y_t \mathbf{x}_t}{\sqrt{\mathbf{x}_t^\top \Sigma_t \mathbf{x}_t}}$ $c_t = cb^\top$, $0 \leq c, b \leq 1$;
11     end
12 end
```

The struct "model" contains the following parameters:

- $\gamma > 0$: the desired classification margin
- $c, b \in [0, 1]$: parameters controlling the memory of online learning

2.2.4 Adaptive Regularization Of Weights (AROW)

Unlike the original CW learning algorithm, the Adaptive Regularization Of Weights (AROW) learning introduces the adaptive regularization of the prediction function when processing each new instance in each learning step, making it more robust than CW to sudden changes of label noise in the learning tasks. In particular, the optimization is formulated as follows:

$$(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) = \arg\min_{\boldsymbol{\mu}, \Sigma} D_{KL}(\mathcal{N}(\boldsymbol{\mu}, \Sigma), \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)) + \frac{1}{2\gamma}\ell^2(\boldsymbol{\mu}; (\mathbf{x}_t, y_t)) + \frac{1}{2\gamma}\mathbf{x}_t{}^\top \Sigma_t \mathbf{x}_t$$

where $\ell^2(\boldsymbol{\mu}; (\mathbf{x}_t, y_t)) = (\max\{0, 1 - y_t(\boldsymbol{\mu} \cdot \mathbf{x}_t)\})^2$ and γ is a regularization parameter.

The struct "model" contains the following parameters:

- $\gamma > 0$: the trade-off between the regularization and the loss

Algorithm 10: AROW: Adaptive Regularization Of Weights.

```
1  Initialize: $\mathbf{w}_1 = 0$, $\Sigma_1 = a\mathbf{I}$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\widehat{y}_t = \mathrm{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$, where $\mathbf{w}_t = \boldsymbol{\mu}_t$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}(\max(0, 1 - y_t \mathbf{w}_t^\top \mathbf{x}_t) > 0)$;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \alpha_t \Sigma_t y_t \mathbf{x}_t$ ,    $\Sigma_{t+1} = \Sigma_t - \beta_t \Sigma_t \mathbf{x}_t {\mathbf{x}_t}^\top \Sigma_t$
10         $\alpha_t = \ell(\boldsymbol{\mu}_t; (\mathbf{x}_t, y_t))\beta_t$,    $\beta_t = \frac{1}{{\mathbf{x}_t}^\top \Sigma_t \mathbf{x}_t + \gamma}$;
11     end
12 end
```

2.2.5 New Adaptive Regularization of Weights (NAROW)

The New Adaptive Regularize Of Weights (NAROW) is an algorithm that interpolates between a second-order algorithm with adaptive-second-order-information, like AROW, and one with fixed-second-order-information. Even the bound is in between these two worlds, the matrix Σ_t is updated only less frequently than AROW, preventing its eigenvalues from growing too much.

Algorithm 11: NAROW: New Adaptive Regularization of Weights.

```
1  Initialize: $\mathbf{w}_1 = 0$, $\Sigma_1 = a\mathbf{I}$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\widehat{y}_t = \mathrm{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$, where $\mathbf{w}_t = \Sigma_t \boldsymbol{\mu}_t$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}(\max(0, 1 - y_t \mathbf{w}_t^\top \mathbf{x}_t) > 0)$;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + y_t \mathbf{x}_t$ ,    $\Sigma_{t+1} = \Sigma_t - \beta_t \Sigma_t \mathbf{x}_t {\mathbf{x}_t}^\top \Sigma_t$
10         $\beta_t = \frac{1}{{\mathbf{x}_t}^\top \Sigma_t \mathbf{x}_t + \gamma_t}$, $\gamma_t = \frac{{\mathbf{x}_t}^\top \Sigma_t \mathbf{x}_t}{b{\mathbf{x}_t}^\top \Sigma_t \mathbf{x}_t - 1}$, when ${\mathbf{x}_t}^\top \Sigma_t \mathbf{x}_t \geq 1/b$ and $\gamma_t = +\infty$;
11     end
12 end
```

The struct "model" contains the following parameters:

- $b > 0$: a threshold for deciding the adaptive update

2.2.6 Normal HERD (NHERD)

The normal herd algorithm is introduced based upon constraining the velocity flow over a distribution of weight vectors. In particular, it is designed to effectively herd a Gaussian weight vector distribution by trading off velocity constraints with a loss function. Formally, the updated is performed as follows:

$$\boldsymbol{\mu}_{t+1} = A_t\boldsymbol{\mu}_t + \mathbf{b}_t \ , \quad \Sigma_{t+1} = A_t\Sigma_t A_t^\top, \quad (A_t, \mathbf{b}_t) = \arg\min_{A,\mathbf{b}} \mathbb{E}_{\mathbf{w}_t\sim\mathcal{N}(\boldsymbol{\mu}_t,\Sigma_t)} C_t(A_t\mathbf{w}_t + \mathbf{b}_t),$$

where

$$C_t(\mathbf{w}) = \left[\frac{1}{2}(\mathbf{w}-\mathbf{w}_t)^\top\Sigma_t^{-1}(\mathbf{w}-\mathbf{w}_t) + C\ell(\mathbf{w};(\mathbf{x}_t,y_t))\right],$$

and $\ell(\mathbf{w};(\mathbf{x}_t,y_t)) = (\max\{0, 1-y_t\mathbf{w}_t^\top\mathbf{x}_t\})^2$.

To analytically solve the update problem, the above optimization is further uniformly relaxed as the following objective

$$\begin{aligned}(\boldsymbol{\mu}_{t+1}, A_t) \quad &= \quad \arg\min_{\boldsymbol{\mu},A} \frac{1}{2}(\boldsymbol{\mu}-\boldsymbol{\mu}_t)^\top\Sigma_t^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_t) + C\ell(\boldsymbol{\mu};(\mathbf{x}_t,y_t)) \\ &\quad + \frac{1}{2}Tr((A-I)^\top\Sigma_t^{-1}(A-I)\Sigma_t) + \frac{C}{2}\mathbf{x}_t^\top A\Sigma_t A^\top\mathbf{x}_t\end{aligned}$$

which enjoy a closed-form solution.

Algorithm 12: NAROW: New Adaptive Regularization of Weights.

```
1  Initialize: $\mathbf{w}_1 = 0$, $\Sigma_1 = a\mathbf{I}$
2  for $t = 1, 2, \dots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\widehat{y}_t = \text{sign}(\mathbf{w}_t^\top\mathbf{x}_t)$, where $\mathbf{w}_t = \boldsymbol{\mu}_t$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}((\max\{0, 1-y_t\mathbf{w}_t^\top\mathbf{x}_t\})^2 > 0)$;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \alpha_t\Sigma_t y_t\mathbf{x}_t$ ,   $\Sigma_{t+1} = \Sigma_t - \beta_t\Sigma_t\mathbf{x}_t\mathbf{x}_t{}^\top\Sigma_t$
10         $\alpha_t = \frac{\max\{0,1-y_t\boldsymbol{\mu}_t^\top\mathbf{x}_t\}}{\mathbf{x}_t^\top\Sigma_t\mathbf{x}_t+1/C}$ ,   $\beta_t = \frac{C^2\mathbf{x}_t\Sigma_t\mathbf{x}_t^\top+2C}{(1+C\mathbf{x}_t\Sigma_t\mathbf{x}_t^\top)^2}$;
11     end
12 end
```

The struct "model" contains the following parameters:

- $C > 0$: a trade-off between regularization and the loss

2.2.7 Soft Confidence-weighted learning (SCW)

This section presents the soft confidence-weighted (SCW) learning [14], which aims to address the limitation of the CW and AROW learning. After the introduction of the following loss function:

$$\ell^{\phi}\big(\mathcal{N}(\boldsymbol{\mu}, \Sigma); (\mathbf{x}_t, y_t)\big) = \max\Big(0, \phi\sqrt{\mathbf{x}_t^\top \Sigma \mathbf{x}_t} - y_t \boldsymbol{\mu} \cdot \mathbf{x}_t\Big),$$

where $\phi = \Phi^{-1}(\eta)$, it is easy to verify that the optimization problem of the original CW can be re-written as follows

$$\begin{aligned}(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) = \arg\min_{\boldsymbol{\mu},\Sigma} D_{KL}\big(\mathcal{N}(\boldsymbol{\mu}, \Sigma) \| \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)\big) \\ s.t.\ \ell^{\phi}\big(\mathcal{N}(\boldsymbol{\mu}, \Sigma); (\mathbf{x}_t, y_t)\big) = 0,\ \phi > 0\end{aligned}$$

The original CW learning method employs a very aggressive updating strategy by changing the distribution as much as necessary to satisfy the constraint imposed by the current example. Although it results in the rapid learning effect, it could force to wrongly change the parameters of the distribution dramatically when handling a mislabeled instance. Such undesirable property makes the original CW algorithm performs poorly in many real-world applications with relatively large noise.

To overcome the above limitation of the CW learning problem, Soft Confidence-Weighted (SCW) learning method is proposed as follows:

$$(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) = \arg\min_{\boldsymbol{\mu},\Sigma} D_{KL}\big(\mathcal{N}(\boldsymbol{\mu}, \Sigma) \| \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)\big) + C\ell^{\phi}\big(\mathcal{N}(\boldsymbol{\mu}, \Sigma); (\mathbf{x}_t, y_t)\big)$$

where C is a parameter to tradeoff the passiveness and aggressiveness. The above formulation of the Soft Confidence-Weighted algorithm is denoted as "SCW-I" for short.

Furthermore, employing a squared penalty leads to the second formulation of SCW learning (denoted as "SCW-II" for short):

$$(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) = \arg\min_{\boldsymbol{\mu},\Sigma} D_{KL}\big(\mathcal{N}(\boldsymbol{\mu}, \Sigma) \| \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t)\big) + C\ell^{\phi}\big(\mathcal{N}(\boldsymbol{\mu}, \Sigma); (\mathbf{x}_t, y_t)\big)^2$$

The struct "model" contains the following parameters:

- $a > 0$, which is used to initialize the $\Sigma_1 = a\mathbf{I}$, where $\mathbf{I}$ is identity matrix;
- $\phi > 0$, which is $\phi = \Phi^{-1}(\eta)$, where η is the probability required for the updated distribution on the current instance
- $C > 0$, which is used to trade-off between keeping the previous information and minimizing the current loss

Algorithm 13: SCW: Soft Confidence Weighted algorithms.

```
1  Initialize: $\mathbf{w}_1 = 0$, $\Sigma_1 = a\mathbf{I}$, $\phi = \Phi^{-1}(\eta)$, $\psi = 1 + \frac{\phi^2}{2}$ and $\zeta = 1 + \phi^2$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\widehat{y}_t = \mathrm{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$, where $\mathbf{w}_t = \boldsymbol{\mu}_t$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}(\max\{0, 1 - y_t \mathbf{w}_t^\top \mathbf{x}_t\} > 0)$;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \alpha_t y_t \Sigma_t \mathbf{x}_t$ ,    $\Sigma_{t+1} = \Sigma_t - \beta_t \Sigma_t \mathbf{x}_t{}^\top \mathbf{x}_t \Sigma_t$
10         $\alpha_t = \begin{cases} \min\{C, \max\{0, \frac{1}{\upsilon_t \zeta}(-m_t \psi + \sqrt{m_t{}^2 \frac{\phi^4}{4} + \upsilon_t \phi^2 \zeta})\}\} & \text{SCW-I,} \\ \max\{0, \frac{-(2m_t n_t + \phi^2 m_t \upsilon_t) + \gamma_t}{2(n_t^2 + n_t \upsilon_t \phi^2)}\} & \text{SCW-II.} \end{cases}$
           $\beta_t = \frac{\alpha_t \phi}{\sqrt{u_t} + \upsilon_t \alpha_t \phi}$, $u_t = \frac{1}{4}(-\alpha_t \upsilon_t \phi + \sqrt{\alpha_t{}^2 \upsilon_t{}^2 \phi^2 + 4\upsilon_t})^2$,
           $\upsilon_t = \mathbf{x}_t{}^\top \Sigma_t \mathbf{x}_t$,$m_t = y_t(\boldsymbol{\mu}_t \cdot \mathbf{x}_t)$, $\gamma_t = \phi\sqrt{\phi^2 m_t^2 \upsilon_t^2 + 4 n_t \upsilon_t (n_t + \upsilon_t \phi^2)}$, and
           $n_t = \upsilon_t + \frac{1}{2C}$.
11     end
12 end
```

3 Online Learning Algorithms for Multiclass Classification

3.1 First-Order Algorithms

In this section, we provide the details of the first-order learning algorithms and the corresponding functions (located at /algorithms/Multiclass-Classification). The first-order learning algorithms only keep updating k classification functions (for the k different labels), which only utilize the first-order information of the examples.

Specifically, the struct "model" in the codes contains a field "W", which is a $k \times d$ matrix, whose i-th row is the linear classifier for the i-th label. In addition, "model" also contains various different parameters for different algorithms, which will be listed in every algorithm.

3.1.1 Multiclass Perceptron

Crammer and Singer extended the binary Perceptron algorithm to a family of multiclass Perceptron algorithms [8], by allocating different weights on the support vectors added to the classifiers corresponding to those error-set $E = \{i \neq y : W_i \cdot \mathbf{x} \geq W_y \cdot \mathbf{x}\}$. According to different allocation strategies, three variants of multiclass Perceptron algorithms are proposed: max-score multiclass Perceptron, uniform multiclass Perceptron, and proportion multiclass Perceptron.

Algorithm 14: M-Perceptron: multiclass Perceptron algorithms.

```
1  Initialize: $\mathbf{W}_1 = 0$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\hat{y}_t = \arg\max_{i=1}^{k}(W_{t,i} \cdot \mathbf{x}_t)$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}(\widehat{y}_t \neq y_t)$.;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $W_{t+1,i} = W_{t,i} + \alpha_{t,i}\mathbf{x}_t$
10     end
11 end
```

For max-score multiclass update,

$$\alpha_{t,i} = \begin{cases} -1 & i = \arg\max_{j=1}^{k} W_{t,j} \cdot \mathbf{x}_t \\ 1 & i = y_t \\ 0 & \text{otherwise} \end{cases}$$

For uniform multiclass update,

$$\alpha_{t,i} = \begin{cases} -1/|E_t| & i \in E_t,\ E_t = \{i \neq y_t : W_{t,i} \cdot \mathbf{x}_t \geq W_{t,y_t} \cdot \mathbf{x}_t\} \\ 1 & i = y_t \\ 0 & \text{otherwise} \end{cases}$$

For promotion multiclass update,

$$\alpha_{t,i} = \begin{cases} -\frac{[W_{t,i}\cdot\mathbf{x}_t - W_{t,y_t}\cdot\mathbf{x}_t]_+}{\sum_{j=1}^{k}[W_{t,j}\cdot\mathbf{x}_t - W_{t,y_t}\cdot\mathbf{x}_t]_+} & i \neq y_t \\ 1 & i = y_t \end{cases}$$

where $[z]_+ = \max(z, 0)$.

3.1.2 Multiclass Relaxed Online Maximum Margin Algorithm

Algorithm 15: M-ROMMA: Multiclass Relaxed Online Maximum Margin Algorithm algorithms.

```
1  Initialize: $\mathbf{W}_1 = 0$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\hat{y}_t = \arg\max_{i=1}^{k}(W_{t,i}\cdot\mathbf{x}_t)$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss:
           $l_t = \begin{cases} \mathbb{I}(\widehat{y}_t \neq y_t) & \text{Multiclass ROMMA} \\ \mathbb{I}(\max(0, 1-(W_{t,y_t}\cdot\mathbf{x}_t - \max_{j\neq y_t} W_{t,j}\cdot\mathbf{x}_t)) > 0) & \text{Multiclass agg-ROMMA} \end{cases}$
       if $l_t > 0$ then
7          The learner updates the classification model:
8          $W_{t+1,i} = c_t W_{t,i} + \alpha_{t,i} d_t \mathbf{x}_t$
9          $c_t = \frac{2\|\mathbf{x}_t\|^2\|W\|^2 - (W_{t,y_t}\cdot\mathbf{x}_t - \max_{j\neq y_t} W_{t,j}\cdot\mathbf{x}_t)}{2\|\mathbf{x}_t\|^2\|W\|^2 - (W_{t,y_t}\cdot\mathbf{x}_t - \max_{j\neq y_t} W_{t,j}\cdot\mathbf{x}_t)^2}$
10         $d_t = \frac{\|W\|^2(1 - (W_{t,y_t}\cdot\mathbf{x}_t - \max_{j\neq y_t} W_{t,j}\cdot\mathbf{x}_t))}{2\|\mathbf{x}_t\|^2\|W\|^2 - (W_{t,y_t}\cdot\mathbf{x}_t - \max_{j\neq y_t} W_{t,j}\cdot\mathbf{x}_t)^2}$
11             $\alpha_{t,i} = \begin{cases} -1 & i = \arg\max_{j=1}^{k} W_{t,j}\cdot\mathbf{x}_t \\ 1 & i = y_t \\ 0 & \text{otherwise} \end{cases}$
12     end
13 end
```

The ROMMA and agg-ROMMA algorithm do not contains parameters.

3.1.3 Multiclass Online Gradient Descent Algorithm

The online gradient descent algorithm [18] applies the gradient descent updating approach together with the following hinge loss:

$$\ell(W; (\mathbf{x}_t, y_t)) = \max\left(0, 1 - (W_{y_t}\cdot\mathbf{x}_t - \max_{j\neq y_t} W_j\cdot\mathbf{x}_t)\right)$$

Algorithm 16: M-OGD: Multiclass Online Gradient Descent Algorithm.

```
1  Initialize: $\mathbf{W}_1 = 0$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\hat{y}_t = \arg\max_{i=1}^{k}(W_{t,i} \cdot \mathbf{x}_t)$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}(\ell(W_t; (\mathbf{x}_t, y_t)) > 0)$;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $W_{t+1,i} = W_{t,i} + \alpha_{t,i}\mathbf{x}_t$
10         $$\alpha_{t,i} = \begin{cases} -1/\sqrt{t} & i = \arg\max_{j=1}^{k} W_{t,j} \cdot \mathbf{x}_t \\ 1/\sqrt{t} & i = y_t \\ 0 & \text{otherwise} \end{cases}$$
11     end
12 end
```

In our implementation, the algorithm does not contains any parameters.

3.1.4 Multiclass Passive Aggressive Algorithms

The family of online multiclass Passive-Aggressive (PA) learning [2] is extended from its binary version, which is formulated to trade off minimizing the distance between the learnt classifier and the previous classifier, and minimizing the loss of the learnt classier suffered on the current instance.

Formally the PA algorithm is formulated as the following online optimization problem

$$W_{t+1} = \arg\min_{W} \frac{1}{2}\|W - W_t\|^2 \ , \quad \text{s.t. } \ell(W; (\mathbf{x}_t, y_t)) = \max\left(0, 1 - (W_{y_t} \cdot \mathbf{x}_t - \max_{j \neq y_t} W_j \cdot \mathbf{x}_t)\right) = 0$$

The multiclass Passive Aggressive algorithm is extended to multiclass Passive Aggressive I and II algorithms, which can better handle the non-separable case. Multiclass Passive Aggressive I is formulated as

$$W_{t+1} = \arg\min_{W} \frac{1}{2}\|W - W_t\|^2 + C\ell(W; (\mathbf{x}_t, y_t))$$

Multiclass Passive Aggressive II is formulated as

$$W_{t+1} = \arg\min_{W} \frac{1}{2}\|W - W_t\|^2 + C\ell(W; (\mathbf{x}_t, y_t))^2$$

For multiclass Passive Aggressive algorithm

$$\alpha_{t,i} = \begin{cases} -\frac{\ell(W_t;(\mathbf{x}_t,y_t))}{2\|\mathbf{x}_t\|^2} & i = \arg\max_{j=1}^{k} W_{t,j} \cdot \mathbf{x}_t \\ \frac{\ell(W_t;(\mathbf{x}_t,y_t))}{2\|\mathbf{x}_t\|^2} & i = y_t \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 17: M-PA: Multiclass Passive Aggressive Algorithms.

```
1  Initialize: $\mathbf{W}_1 = 0$, $C > 0$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\hat{y}_t = \arg\max_{i=1}^{k}(W_{t,i} \cdot \mathbf{x}_t)$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner calculates the suffered loss: $l_t = \mathbb{I}(\ell(W_t; (\mathbf{x}_t, y_t)) > 0)$;
7      if $l_t > 0$ then
8          The learner updates the classification model:
9          $W_{t+1,i} = W_{t,i} + \alpha_{t,i}\mathbf{x}_t$;
10     end
11 end
```

For multiclass Passive Aggressive I algorithm

$$\alpha_{t,i} = \begin{cases} -\min(C, \frac{\ell(W_t;(\mathbf{x}_t,y_t))}{2\|\mathbf{x}_t\|^2}) & i = \arg\max_{j=1}^{k} W_{t,j} \cdot \mathbf{x}_t \\ \min(C, \frac{\ell(W_t;(\mathbf{x}_t,y_t))}{2\|\mathbf{x}_t\|^2}) & i = y_t \\ 0 & \text{otherwise} \end{cases}$$

For multiclass Passive Aggressive II algorithm, the function

$$\alpha_{t,i} = \begin{cases} -\frac{\ell(W_t;(\mathbf{x}_t,y_t))}{2\|\mathbf{x}_t\|^2 + \frac{1}{2C}} & i = \arg\max_{j=1}^{k} W_{t,j} \cdot \mathbf{x}_t \\ \frac{\ell(W_t;(\mathbf{x}_t,y_t))}{2\|\mathbf{x}_t\|^2 + \frac{1}{2C}} & i = y_t \\ 0 & \text{otherwise} \end{cases}$$

3.2 Second-Order Algorithms

To better exploring the underlying structure between features, the second-order online learning algorithms typically assume the weight vector follows Gaussian distributions $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ with mean vector $\boldsymbol{\mu} \in \mathbb{R}^{kd}$ and covariance matrix $\Sigma \in \mathbb{R}^{kd \times kd}$.

Specifically, the struct "model" in the codes contains two fields "$\boldsymbol{\mu} \in \mathbb{R}^{kd}$" and "$\Sigma \in \mathbb{R}^{kd \times kd}$", which are the mean vector and covariance matrix for the learnt Gaussian distribution. In addition, "model" also contains various parameters for different algorithms, which will be explained in every algorithm.

3.2.1 Multiclass Confidence Weighted Learning

In multiclass confidence-weighted learning [3], the weight distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ is updated with one strategy similar with the binary case. To make the proposed algorithms more efficient, the authors proposed three variant of multiclass confidence weighted algorithms. Here, we implemented the most efficient one.

Specifically, we introduced a new label-dependent feature, $\psi(\mathbf{x}, i) = [\mathbf{0}^\top, \ldots, \mathbf{x}^\top, \ldots, \mathbf{0}^\top]^\top$, where $\mathbf{0}, \mathbf{x} \in \mathbb{R}^d$, only the i-th position of $\psi(\mathbf{x}, i)$ is $\mathbf{x}$ and the others are $\mathbf{0}$.

Algorithm 18: M-CW: Multiclass Confidence Weighted Algorithms.

```
1  Initialize: $\boldsymbol{\mu}_1 = \mathbf{0}$" and "$\Sigma_1 = \mathbf{I}$, $\phi = \Phi^{-1}(\eta)$ and $\eta \in (0.5, 1]$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\hat{y}_t = \arg\max_{i=1}^{k}(\mathbf{w}_t \cdot \psi(\mathbf{x}_t, i))$, where
       $\mathbf{w}_t = \boldsymbol{\mu}_t$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner computes all the updating coefficients:
7      $m_t = \boldsymbol{\mu}_t^\top \Delta\psi_t$, $v_t = \Delta\psi_t{}^\top \Sigma_t \Delta\psi_t$
8      $\alpha_t = \max\{0, \frac{-(1+2\phi m_t)+\sqrt{(1+2\phi m_t)^2 - 8\phi(m_t - \phi v_t)}}{4\phi v_t}\}$,   $\beta_t = \frac{1}{1/(2\alpha_t \phi) + v_t}$;
9      The learner calculates the suffered loss: $l_t = \mathbb{I}(\alpha_t > 0)$;
10     if $l_t > 0$ then
11         The learner updates the classification model:
12         $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \alpha_t \Sigma_t \Delta\psi_t$ ,   $\Sigma_{t+1} = \Sigma_t - \beta_t \Sigma_t \Delta\psi_t \Delta\psi_t{}^\top \Sigma_t$
13         where $\Delta\psi_t = \psi(\mathbf{x}_t, y_t) - \psi(\mathbf{x}_t, \widetilde{y}_t)$, $\widetilde{y}_t = \arg\max_{i=1, i \neq y_t}^{k}(\boldsymbol{\mu}_i \cdot \mathbf{x}_t)$;
14     end
15 end
```

The struct "model" contains the following parameters:

- $\eta \in (0.5, 1]$, which is the probability required for the updated distribution on the current instance

3.2.2 Multiclass Adaptive Regularization Of Weights

The multiclass AROW [6] is a natural extension of of binary AROW algorithm. To extend the binary algorithm, the authors proposed two kinds of updating methods. Here, the implemented version is the most efficient one.

Algorithm 19: M-AROW: Multiclass Adaptive Regularization Of Weights.

```
1  Initialize: $\boldsymbol{\mu}_1 = \mathbf{0}$” and “$\Sigma_1 = \mathbf{I}$, $\phi = \Phi^{-1}(\eta)$, $\eta \in (0.5, 1]$, and $r > 0$
2  for $t = 1, 2, \ldots, T$ do
3      The learner receives an incoming instance: $\mathbf{x}_t \in \mathcal{X}$;
4      The learner predicts the class label: $\hat{y}_t = \arg\max_{i=1}^{k}(\mathbf{w}_t \cdot \psi(\mathbf{x}_t, i))$, where
       $\mathbf{w}_t = \boldsymbol{\mu}_t$;
5      The true class label is revealed from the environment: $y_t \in \mathcal{Y}$;
6      The learner computes all the updating coefficients:
7      $m_t = \boldsymbol{\mu}_t^\top \Delta\psi_t$, $v_t = \Delta\psi_t{}^\top \Sigma_t \Delta\psi_t$, $\alpha_t = \max\{0, 1 - m_t\}\beta_t$,   $\beta_t = \frac{1}{r + v_t}$;
8      The learner calculates the suffered loss: $l_t = \mathbb{I}(m_t < 1)$;
9      if $l_t > 0$ then
10         The learner updates the classification model:
11         $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \alpha_t \Sigma_t \Delta\psi_t$ ,   $\Sigma_{t+1} = \Sigma_t - \beta_t \Sigma_t \Delta\psi_t \Delta\psi_t{}^\top \Sigma_t$
12         where $\Delta\psi_t = \psi(\mathbf{x}_t, y_t) - \psi(\mathbf{x}_t, \widetilde{y}_t)$, $\widetilde{y}_t = \arg\max_{i=1, i \neq y_t}^{k}(\boldsymbol{\mu}_i \cdot \mathbf{x}_t)$;
13     end
14 end
```

The struct “model” contains the following parameters:

- $r \in (0, +\infty)$,

3.2.3 Multiclass Soft Confidence-Weighted Learning

In Multiclass Soft Confidence-Weighted learning, we assume the each prototype vector $\mathbf{w}_i$ follows the Gaussian distribution with the mean vector $\boldsymbol{\mu}_i$ and the covariance matrix Σ_i. For simplicity, we assume each prototype $i \in [1, k]$ share the same covariance matrix Σ. In multiclass learning setting, we want to make that the probability of the score of the class of the smallest score among all relevant classes to be higher than the score of the class of the highest score among all irrelevant classes is larger than a threshold η. In mathematical form, the constrain can be expressed as follows:

$$Pr_{\mathbf{w}_{t,r_t} \sim \mathcal{N}(\boldsymbol{\mu}_{t,r_t}, \Sigma_t), \mathbf{w}_{t,s_t} \sim \mathcal{N}(\boldsymbol{\mu}_{t,s_t}, \Sigma_t)}[(\mathbf{w}_{t,r_t} \cdot \mathbf{x}_t) \geq (\mathbf{w}_{t,s_t} \cdot \mathbf{x}_t)] \geq \eta \tag{1}$$

where

$$r_t = \arg\min_{r \in Y_t} \boldsymbol{\mu}_{t,r} \cdot x_t, \quad s_t = \arg\max_{s \notin Y_t} \boldsymbol{\mu}_{t,s} \cdot x_t \tag{2}$$

To overcome the above limitation of the CW learning problem, we propose a Soft Confidence-Weighted (SCW) learning method, which aims to soften the aggressiveness of the CW updating strategy. The idea of the SCW learning is inspired by the variants of PA algorithms (PA-I and PA-II) and the adaptive margin. In particular, we formulate the optimization of SCW for learning the soft-margin classifiers as follows:

$$\begin{aligned}
(\boldsymbol{\mu}_{t+1,r_t}, \boldsymbol{\mu}_{t+1,s_t}, \Sigma_{t+1}) &= \arg\min_{\boldsymbol{\mu}_r, \boldsymbol{\mu}_s, \Sigma} D_{KL}\big(\mathcal{N}(\boldsymbol{\mu}_r, \Sigma) \| \mathcal{N}(\boldsymbol{\mu}_{t,r_t}, \Sigma_t)\big) && (3)\\
&+ D_{KL}\big(\mathcal{N}(\boldsymbol{\mu}_s, \Sigma) \| \mathcal{N}(\boldsymbol{\mu}_{t,s_t}, \Sigma_t)\big) && (4)\\
&+ C\ell^{\phi}\big(\mathcal{N}(\boldsymbol{\mu}_r, \boldsymbol{\mu}_s, \Sigma_t); (\mathbf{x}_t, y_t)\big) && (5)
\end{aligned}$$

where C is a parameter to tradeoff the passiveness and aggressiveness. We denoted the above formulation of the Multiclass Soft Confidence-Weighted algorithm, as "M-SCW1" for short.

The closed-form solution of the optimization (3) is expressed as:

$$\begin{aligned}
\boldsymbol{\mu}_{t+1,r_t} &= \boldsymbol{\mu}_{t,r_t} + \alpha_t y_t \Sigma_t \mathbf{x}_t, \quad \boldsymbol{\mu}_{t+1,s_t} = \boldsymbol{\mu}_{t,s_t} - \alpha_t y_t \Sigma_t \mathbf{x}_t,\\
\Sigma_{t+1} &= \Sigma_t - \beta_t \Sigma_t {\mathbf{x}_t}^T \mathbf{x}_t \Sigma_t
\end{aligned}$$

where the updating coefficients are as follows:

$$\begin{aligned}
\alpha_t &= \min\{C, \max\{0, \frac{1}{2\upsilon_t \psi}(-m_t \psi + \sqrt{{m_t}^2 \psi^2 - {m_t}^2 \psi + 2\psi \phi^2 \upsilon_t})\}\}\\
\beta_t &= \frac{\alpha_t \phi}{\sqrt{2u_t} + \upsilon_t \alpha_t \phi}
\end{aligned}$$

where $u_t = \frac{1}{8}(-\alpha_t \upsilon_t \phi + \sqrt{{\alpha_t}^2 {\upsilon_t}^2 \phi^2 + 8\upsilon_t})^2$,$\upsilon_t = {\mathbf{x}_t}^T \Sigma_t \mathbf{x}_t$,$m_t = \boldsymbol{\mu}_{t,r_t} \cdot \mathbf{x}_t - \boldsymbol{\mu}_{t,s_t} \cdot \mathbf{x}_t$,$\phi = \Phi^{-1}(\eta)$,$\psi = 1 + \frac{\phi^2}{2}$.

Similar to the variant of PA, we can also modify the above formulation by employing a squared penalty, leading to the second formulation of Multiclass SCW learning (denoted as "M-SCW2" for short):

$$\begin{aligned}
(\boldsymbol{\mu}_{t+1,r_t}, \boldsymbol{\mu}_{t+1,s_t}, \Sigma_{t+1}) &= \arg\min_{\boldsymbol{\mu}_r, \boldsymbol{\mu}_s, \Sigma} D_{KL}\big(\mathcal{N}(\boldsymbol{\mu}_r, \Sigma) \| \mathcal{N}(\boldsymbol{\mu}_{t,r_t}, \Sigma_t)\big) && (6)\\
&+ D_{KL}\big(\mathcal{N}(\boldsymbol{\mu}_s, \Sigma) \| \mathcal{N}(\boldsymbol{\mu}_{t,s_t}, \Sigma_t)\big) && (7)\\
&+ C\ell^{\phi}\big(\mathcal{N}(\boldsymbol{\mu}_r, \boldsymbol{\mu}_s, \Sigma_t); (\mathbf{x}_t, y_t)\big)^2 && (8)
\end{aligned}$$

The closed-form solution of the optimization (6) is:

$$\begin{aligned}
\boldsymbol{\mu}_{t+1,r_t} &= \boldsymbol{\mu}_{t,r_t} + \alpha_t y_t \Sigma_t \mathbf{x}_t, \quad \boldsymbol{\mu}_{t+1,s_t} = \boldsymbol{\mu}_{t,s_t} - \alpha_t y_t \Sigma_t \mathbf{x}_t,\\
\Sigma_{t+1} &= \Sigma_t - \beta_t \Sigma_t {\mathbf{x}_t}^T \mathbf{x}_t \Sigma_t
\end{aligned}$$

The updating coefficients are as follows:

$$\alpha_t = \max\{0, \frac{-(2m_t \rho_t + \phi^2 m_t \upsilon_t) + \gamma_t}{2(\rho_t^2 + \rho_t \upsilon_t \phi^2)}\}, \quad \beta_t = \frac{\alpha_t \phi}{\sqrt{2u_t} + \upsilon_t \alpha_t \phi}$$

where $\gamma_t = \phi\sqrt{\phi^2 m_t^2 \upsilon_t^2 + 8\rho_t \upsilon_t(\rho_t + \upsilon_t \phi^2)}$, and $\rho_t = 2\upsilon_t + \frac{1}{2C}$.

In the implementation of MSCW1 an MSCW2 functions, you can specify the struct "model" with the following parameters:

- $a > 0$, which is used to initialize the $\Sigma_1 = a\mathbf{I}$, where $\mathbf{I}$ is identity matrix;
- $\phi > 0$, which is $\phi = \Phi^{-1}(\eta)$, where η is the probability required for the updated distribution on the current instance
- $C > 0$, which is used to trade-off between keeping the previous information and minimizing the current loss

Algorithm 20: Multiclass SCW learning algorithms (**M-SCW**)

```
1  INPUT: parameters $C > 0$, $\eta > 0$.
2  INITIALIZATION: $\boldsymbol{\mu}_{1,1}, ..., \boldsymbol{\mu}_{1,k} = (0, \dots, 0)^\top$, $\Sigma_1 = I$. for $t = 1, \dots, T$ do
3      Receive an example $\mathbf{x}_t \in \mathbb{R}^d$;
4      Make prediction: $\hat{Y}_t = \arg\max_r(\boldsymbol{\mu}_{t,r} \cdot \mathbf{x}_t), r \in [1, k]$;
5      Receive true label $Y_t$;
6      Suffer loss $\ell^\phi\big(\mathcal{N}(\boldsymbol{\mu}_{t,r_t}, \boldsymbol{\mu}_{t,s_t}, \Sigma_t); (\mathbf{x}_t, Y_t)\big)$;
7      if $\ell^\phi\big(\mathcal{N}(\boldsymbol{\mu}_{t,r_t}, \boldsymbol{\mu}_{t,s_t}, \Sigma_t); (\mathbf{x}_t, Y_t)\big) > 0$ then
8          $\boldsymbol{\mu}_{t+1,r_t} = \boldsymbol{\mu}_{t,r_t} + \alpha_t y_t \Sigma_t \mathbf{x}_t$,
9          $\boldsymbol{\mu}_{t+1,s_t} = \boldsymbol{\mu}_{t,s_t} - \alpha_t y_t \Sigma_t \mathbf{x}_t$,
10         $\Sigma_{t+1} = \Sigma_t - \beta_t \Sigma_t {\mathbf{x}_t}^T \mathbf{x}_t \Sigma_t$
11         where $\alpha_t$ and $\beta_t$ are computed as follows:

           $\text{MSCW1} = \begin{cases} \alpha_t = \min\{C, \max\{0, \frac{1}{2\upsilon_t\psi}(-m_t\psi + \sqrt{{m_t}^2\psi^2 - {m_t}^2\psi + 2\psi\phi^2\upsilon_t})\}\} \\ \beta_t = \frac{\alpha_t\phi}{\sqrt{2u_t} + \upsilon_t\alpha_t\phi} \end{cases}$

           where $u_t = \frac{1}{8}(-\alpha_t\upsilon_t\phi + \sqrt{{\alpha_t}^2{\upsilon_t}^2\phi^2 + 8\upsilon_t})^2$,$\upsilon_t = {\mathbf{x}_t}^T\Sigma_t\mathbf{x}_t$,$m_t =$
           $\boldsymbol{\mu}_{t,r_t} \cdot \mathbf{x}_t - \boldsymbol{\mu}_{t,s_t} \cdot \mathbf{x}_t$,$\phi = \Phi^{-1}(\eta)$,$\psi = 1 + \frac{\phi^2}{2}$.

           $\text{MSCW2} = \begin{cases} \alpha_t = \max\{0, \frac{-(2m_t\rho_t + \phi^2 m_t\upsilon_t) + \gamma_t}{2(\rho_t^2 + \rho_t\upsilon_t\phi^2)}\} \\ \beta_t = \frac{\alpha_t\phi}{\sqrt{2u_t} + \upsilon_t\alpha_t\phi} \end{cases}$

           where $\gamma_t = \phi\sqrt{\phi^2 m_t^2 \upsilon_t^2 + 8\rho_t\upsilon_t(\rho_t + \upsilon_t\phi^2)}$, and $\rho_t = 2\upsilon_t + \frac{1}{2C}$.
12     end
13 end
```

4 Conclusions, Revision, and Citation

LIBOL is an easy-to-use open source package for efficient and scalable on-line linear classification. It is currently one of comprehensive online learning software packages that include the largest number of diverse online learning algorithms for online classification. LIBOL is still being improved by improvements from practical users and new research results [16, 17]. The ultimate goal is to make easy learning with massive-scale data streams to tackle the emerging grand challenge of big data mining.

Revision History

- Version 0.1 was released on 28 December 2012. This version mainly includes MATLAB implementation for binary classification .
- Version 0.2 was released on 28 July 2013. This version includes several major improvements: (i) C++ implementations for the main functions; (ii) online learning algorithms for multi-class classification; (iii) new design and framework.

More details of the revision history can be found in LIBOL website:

`http://LIBOL.stevenhoi.org/changelog.html`

Welcome to send us your suggestions/corrections for LIBOL by the following email:

`chhoi@ntu.edu.sg`

Citation

In citing LIBOL in your papers, please use the following reference:

S.C.H. Hoi, J. Wang, P. Zhao. LIBOL: A Library for Online Learning Algorithms. Nanyang Technological University, 2012. `http://LIBOL.stevenhoi.org`.

References

[1] N. Cesa-Bianchi, A. Conconi, and C. Gentile. A second-order perceptron algorithm. *SIAM J. Comput.*, 34(3):640–668, 2005.

[2] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[3] K. Crammer, M. Dredze, and A. Kulesza. Multi-class confidence weighted algorithms. In *EMNLP*, pages 496–504, 2009.

[4] K. Crammer, M. Dredze, and F. Pereira. Exact convex confidence-weighted learning. In *NIPS*, pages 345–352, 2008.

[5] K. Crammer, A. Kulesza, and M. Dredze. Adaptive regularization of weight vectors. In *NIPS*, pages 414–422, 2009.

[6] K. Crammer, A. Kulesza, and M. Dredze. Adaptive regularization of weight vectors. *Machine Learning*, 91(2):155–187, 2013.

[7] K. Crammer and D. D. Lee. Learning via gaussian herding. In *NIPS*, pages 451–459, 2010.

[8] K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, 2003.

[9] M. Fink, S. Shalev-Shwartz, Y. Singer, and S. Ullman. Online multiclass learning by interclass hypothesis sharing. In *Proceedings of the 25th International Conference on Machine learning (ICML'06)*, pages 313–320, 2006.

[10] C. Gentile. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, 2001.

[11] Y. Li and P. M. Long. The relaxed online maximum margin algorithm. *Machine Learning*, 46(1-3):361–387, 2002.

[12] F. Orabona and K. Crammer. New adaptive algorithms for online classification. In *NIPS*, pages 1840–1848, 2010.

[13] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psych. Rev.*, 7:551–585, 1958.

[14] J. Wang, P. Zhao, and S. C. H. Hoi. Exact soft confidence-weighted learning. In *ICML*, 2012.

[15] L. Yang, R. Jin, and J. Ye. Online learning by ellipsoid method. In *ICML*, page 145, 2009.

[16] P. Zhao, S. C. H. Hoi, and R. Jin. Double updating online learning. *Journal of Machine Learning Research*, 12:1587–1615, 2011.

[17] P. Zhao, S. C. H. Hoi, R. Jin, and T. Yang. Online auc maximization. In *ICML*, pages 233–240, 2011.

[18] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, pages 928–936, 2003.