© Xinghua Lou (xinghua.lou@gmail.com)
September 11, 2012

PLEASD: A Matlab Toolbox for Structured Learning

1 Introduction

PLEASD stands for **P**rediction and **LEA**rning for **S**tructured **D**ata. It is a Matlab toolbox of algorithmic frameworks for training structured prediction models. Unlike flat data to which many machine learning algorithms can be applied as a black-box, structured learning usually requires problem-specific formulation and solver implementation. Therefore, because of the huge variety of structured data, there are no generic, or ”black-box” type algorithms that are immediately applicable to a new structured learning problem. Yet, some frameworks can be adapted to work such as max-margin structured SVM and structured perceptron, provided that the problem-specific computation can be treated as plugin functions.

We provide this toolbox to ease the process of applying structured learning to new problems. We attempt to minimize users' involvement in coding the structured learning framework such that they can focus on issues related to their specific problems. To use this toolbox, users simply have to plugin their data and problem-specific functions, subject to very limited interface constraints. Such a plugin based mechanism also allows easy switch between different structured learning algorithms. Currently, PLEASD has included the following structured learning frameworks:

- Bundle method for risk minimization [7];
- Structured perceptron learning [3];
- Structured learning from partial annotations [2];
- Structured perceptron learning from partial annotations [4].

Preliminary: It is best if users can first get familiarized with structured learning including its concept, algorithms and applications [6, 8]. This report does **not** contain detailed introduction to structured learning.

2 Structured Learning in a Nutshell

Briefly speaking, we have a structured input space $\mathcal{X}$ (e.g. sentence, gene sequence) and correspondingly a structured output space $\mathcal{Y}$ (e.g. parsing tree, sequence labeling), and we want to learn a predictor $f : \mathcal{X} \to \mathcal{Y}$ by parameterizing a discriminative function $F : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ with w such that, for given structured input x, we have the optimal structured output

$$y = f(x; w) = \arg\max_{y \in \mathcal{Y}} \{F(x, y; w) := \langle \Psi(x, y), w \rangle\}, \tag{1}$$

where $\Psi(x, y)$ is the so-called joint feature map. $F(x, y; w)$ is then a compatibility score of the input and output pair. To this end, structured learning aims at finding the optimal parameter w that minimizes the sum of loss on given training examples $\{(x_1, \dot{y}_1), \ldots, (x_n, \dot{y}_n)\} \in \mathcal{X} \times \mathcal{Y}$, possibly subject to some regularization on the parameter w. Depending on the problem setting, the learning objective is generally alike to one of the following formulas.

- Structured learning from complete annotations [6, 8]:

$$\min_w \quad \frac{1}{2}\|w\|^2 + \frac{1}{N}\sum_n \left\{ \underbrace{\max_{\hat{y}_n \in \mathcal{Y}_n} [\langle \overbrace{\Psi(x_n, \hat{y}_n)}^{\text{joint feature}}, w \rangle + \overbrace{\Delta(\dot{y}_n, \hat{y}_n)}^{\text{loss}}]}_{\text{predictor}} - \langle \overbrace{\Psi(x_n, \dot{y}_n)}^{\text{joint feature}}, w \rangle \right\}$$

  If the regularization term $\frac{1}{2}\|w\|^2$ is discarded, this formulation turns to structured perceptron [3].

- Structured learning from partial annotations [2]:

$$\min_w \quad \frac{1}{2}\|w\|^2 + \frac{1}{N}\sum_n \left\{ \underbrace{\max_{\hat{y}_n \in \mathcal{Y}_n^P} [\langle \overbrace{\Psi(x_n, \hat{y}_n)}^{\text{joint feature}}, w \rangle + \overbrace{\Delta(\dot{y}_n, \hat{y}_n)}^{\text{loss}}]}_{\text{predictor}} - \underbrace{\max_{\tilde{y}_n \in \mathcal{Y}_n^R} [\langle \overbrace{\Psi(x_n, \tilde{y}_n)}^{\text{joint feature}}, w \rangle]}_{\text{predictor}} \right\}$$

  Similarly, discarding the regularization term $\frac{1}{2}\|w\|^2$ gives structured perceptron from partial annotations [4].

Apparently, we have the following problem-specific parts.

1. **Joint feature** is a feature vector that describes the joint compatibility of the input/output pair (x, y).
2. **Loss** is a scalar that represents the discrepancy between some prediction $\hat{y}$ and the ground truth $\dot{y}$.
3. **Predictor** is the core algorithm for the problem. It solves the inference problem – finding the best structured output given some parameter w. It should also support augmented inference, i.e. the original compatibility scoring function plus a loss term [8]. In the more complicated structured learning from partial annotations, it should also support inference with restricted search space (e.g. space that is consistent with the partial annotation), see [2] for more details.

3 Get Started with PLEASD

3.1 Installation

Installation of the package is very straight forward: download the toolbox from `http://mloss.org/revision/view/1102/`, extract the content of the zip file and add the path to your Matlab.

3.2 Prerequisite

In addition to the Matlab environment, PLEASD requires a quadratic program (QP) solver for updating the learned parameters. You **must** have either the Matlab Optimization Toolbox[1] for function **quadprog** or IBM ILOG CPLEX Optimizer[2] for an alternative **cplexqp**. PLEASD will first try to access **cplexqp** before trying **quadprog**. An error message will appear if none of them exists.

3.3 Data Structure and Data Preparation

The data structure in PLEASD is simply a **cell** of **struct**. In detail, each training data sample is a Matlab **struct** object (say, **d**) which must have two fields: **d.x** as the input and **d.y_dot** as the annotation (output). In addition, some field names are reserved for use in intermediate computation: **d.y_hat** as the predicted output and **d.y_tilde** as a second predicted output used in structured learning partial annotations. Note that PLEASD does not need to know the exact data structure of each field. You can use any data structure for your input and annotation, but only make sure that they can be interpreted by those plugin functions you have to provide (see below). Furthermore, you can tag on any additional information to the **struct**. To this end, the entire training set is simply a **cell** of training data samples.

3.4 Filling Problem-Specific Functions

All problem-specific computation in the learning process can be dissected into three functions: **joint feature**, **loss** and **predictor**. Users must provide those functions according to the following interfaces.

Code 1. *Functor interface for joint feature:*

```
function Psi = myJointFeature(x, y)
% Psi = myJointFeature(x, y) return a vector of features that
% describes the compatibility of input x and output y
%
% Input:
%       x:          structured intput
%       y:          structured output
%
% Output:
%       Psi:        must be a double vector of Dx1 where D is the size of
```

[1]www.mathworks.com/products/optimization/
[2]www.ibm.com/software/integration/optimization/cplex-optimizer/

```
%                       the joint feature vector
%
```

Code 2. *Functor interface for loss:*

```
function res = myLoss(y_dot, y_hat)
% res = myLoss(y_dot, y_hat) returns a scalar that represents the
% difference between the ground truth y_dot and some output y_hat
%
% Input:
%         y_dot:        ground truth output
%         y_hat:        predicted output
%
% Output:
%         res:          a scalar
%
```

Code 3. *Functor interface for predictor:*

```
function y_out = myPredictor(d, w, augment, search_space)
% y_out = myPredictor(d, w, augment, search_space) predicts the most
% likely output y_out for sample d and parameter w.
%
% Input:
%         d:            data sample
%         w:            model parameter
%         augment:      =  0, no loss augment, i.e. <Psi, w> only
%                       =  1, loss augment, i.e. <Psi, w> + Delta(y_dot, y_hat)
%                       = -1, loss augment, i.e. <Psi, w> - Delta(y_dot, y_hat)
%         search_space:     =  0, entire output space
%                           =  1, space consistent with y_dot (partially annotated)
%                           = -1, space inconsistent with y_dot (partially annotated)
%
% Output:
%         y_out:        predicted output
%
```

3.5 Parameter Settings

- Settings for bundle method for risk minimization [7]:

Code 4. *Settings for bundle method for risk minimization:*

```
settings.lambda = 1;        % L2 regularization strength
settings.augment = 1;       % specify augment problem
                    % =  0: no augment
                    % =  1: loss augment, i.e. <Psi, w> + Delta(y_dot, y_hat)
                    % = -1, loss augment, i.e. <Psi, w> - Delta(y_dot, y_hat)
settings.search_space = 0;  % specify search space
                    % =  0, entire output space
                    % =  1, space consistent with y_dot (partially annotated)
                    % = -1, space inconsistent with y_dot (partially annotated)
settings.epsilon = 1e-6;    % convergence condition (approximation gap)
settings.max_iter = 250;    % maximum number of iterations
settings.verbose = 1;       % verbose output
```

- Settings for structured perceptron learning [3]:

Code 5. *Settings for structured perceptron learning:*

```
settings.augment = 1;       % specify augment problem
                    % =  0: no augment
                    % =  1: loss augment, i.e. <Psi, w> + Delta(y_dot, y_hat)
                    % = -1, loss augment, i.e. <Psi, w> - Delta(y_dot, y_hat)
settings.search_space = 0;  % specify search space
                    % =  0, entire output space
                    % =  1, space consistent with y_dot (partially annotated)
                    % = -1, space inconsistent with y_dot (partially annotated)
settings.eta = 1e-5;        % convergence condition (change of w)
settings.rho = 1e-5;        % convergence condition (training error)
settings.max_iter = 250;    % maximum number of iterations
settings.verbose = 1;       % verbose output
```

- Settings for structured learning from partial annotations [2]:

Code 6. *Settings for structured learning from partial annotations:*

```
settings.reward_augment = 0;        % specify augment problem for reward
                    % =  0: no augment
                    % =  1: loss augment, i.e. <Psi, w> + Delta(y_dot, y_hat)
                    % = -1, loss augment, i.e. <Psi, w> - Delta(y_dot, y_hat)
settings.reward_search_space = 1;  % specify search space for reward
                    % =  0, entire output space
```

```
                         % =  1, space consistent with y_dot (partially annotated)
                         % = −1, space inconsistent with y_dot (partially annotated)
settings.penalty_augment = 1;         % specify augment problem for penalty
                         % =  0: no augment
                         % =  1: loss augment, i.e. <Psi, w> + Delta(y_dot, y_hat)
                         % = −1, loss augment, i.e. <Psi, w> − Delta(y_dot, y_hat)
settings.penalty_search_space = 0;  % specify search space for penalty
                         % =  0, entire output space
                         % =  1, space consistent with y_dot (partially annotated)
                         % = −1, space inconsistent with y_dot (partially annotated)
settings.lambda = 1;                  % L2 regularization strength
settings.eta = 1e−5;                  % covergence condition (change of w)
settings.epsilon = 1e−5;              % covergence condition of bmrm
settings.max_iter = 250;              % maximum number of iterations
settings.verbose = 1;                 % verbose output
settings.reuse_cuts = 1;              % reuse cuts
settings.adaptive_epsilons = ...      % adaptive precision
    max((1/2).^(0:settings.max_iter −1), settings.epsilon);
```

- Settings for structured perceptron learning from partial annotations [4]:

Code 7. *Settings for structured perceptron learning from partial annotations:*

```
settings.reward_augment = 0;          % specify augment problem for reward
                         % =  0: no augment
                         % =  1: loss augment, i.e. <Psi, w> + Delta(y_dot, y_hat)
                         % = −1, loss augment, i.e. <Psi, w> − Delta(y_dot, y_hat)
settings.reward_search_space = 1;   % specify search space for reward
                         % =  0, entire output space
                         % =  1, space consistent with y_dot (partially annotated)
                         % = −1, space inconsistent with y_dot (partially annotated)
settings.penalty_augment = 1;         % specify augment problem for penalty
                         % =  0: no augment
                         % =  1: loss augment, i.e. <Psi, w> + Delta(y_dot, y_hat)
                         % = −1, loss augment, i.e. <Psi, w> − Delta(y_dot, y_hat)
settings.penalty_search_space = 0;  % specify search space for penalty
                         % =  0, entire output space
                         % =  1, space consistent with y_dot (partially annotated)
                         % = −1, space inconsistent with y_dot (partially annotated)
settings.eta = 1e−5;                  % covergence condition (change of w)
settings.rho = 1e−5;                  % covergence condition (training error)
settings.max_iter = 250;              % maximum number of iterations
settings.verbose = 1;                 % verbose output
```

4 Demos: Structured Learning for Cell Tracking

Provided that you have prepared your data, implemented those plugin functions and set the runtime parameters, perform a structured learning run is as easy as in the following examples. Users can refer to [1] for more details on this application. Here, the predictor requires a binary integer linear program (binary ILP) solver. Similarly, you **must** have either the Matlab Optimization Toolbox[3] for function **bintprog** or IBM ILOG CPLEX Optimizer[4] for an alternative **cplexbilp** (prior choice by PLEASD). Note that, based on our experience, **cplexbilp** is orders of magnitude faster than **bintprog**. Find all examples in **trackingDemo.m**.

4.1 Demo I: Bundle Method for Risk Minimization

Code 8. *Structured learning for cell tracking based on bundle method for risk minimization (full annotation):*

```
load('data/data-tracking-training-test.mat');

clear settings functors;

% Settings
settings.lambda = 1;          % L2 regularization strength
settings.augment = 1;         % specify augment problem
settings.search_space = 0;    % specify search space
settings.epsilon = 1e-6;      % convergence condition (approximation gap)
settings.max_iter = 250;      % maximum number of iterations
settings.verbose = 1;         % verbose output

% User provided functors
functors.loss = @trackingLoss;
functors.joint_feature = @trackingJointFeature;
functors.predictor = @trackingPredictor;

% Initialize: get the first prediction
w = zeros(sum(colsInCell(Dtraining{1}.x)), 1);
for n = 1:length(Dtraining)
    d = Dtraining{n};
    d.y_dot = d.y_full;
    d.y_hat = trackingPredictor(d, w, settings.augment, settings.search_space);
    Dtraining(n) = {d};
end

ticID = tic;
[w, Dtraining, A, B, R, W, cccp_meta] = bmrm(w, Dtraining, functors, settings);
println('\nTotal␣training␣time:␣%g', toc(ticID));
```

[3]www.mathworks.com/products/optimization/
[4]www.ibm.com/software/integration/optimization/cplex-optimizer/

4.2 Demo II: Structured Learning from Partial Annotations

This demo is the same application as above but uses only partially annotated data samples. More details on structured learning from partial annotations is available at [2].

Code 9. *Structured learning for cell tracking based on structured learning from partial annotations:*

```
load('data/data-tracking-training-test.mat');

clear settings functors;

% Settings
settings.reward_augment = 0;          % specify augment problem for reward
settings.reward_search_space = 1;     % specify search space for reward
settings.penalty_augment = 1;         % specify augment problem for penalty
settings.penalty_search_space = 0;    % specify search space for penalty
settings.lambda = 1;                  % L2 regularization strength
settings.eta = 1e-5;                  % covergence condition (change of w)
settings.epsilon = 1e-5;              % covergence condition of bmrm
settings.max_iter = 250;              % maximum number of iterations
settings.verbose = 1;                 % verbose output
settings.reuse_cuts = 1;              % reuse cuts
settings.adaptive_epsilons = ...      % adaptive precision
    max((1/2).^(0:settings.max_iter-1), settings.epsilon);

% User provided functors
functors.loss = @trackingLoss;
functors.joint_feature = @trackingJointFeature;
functors.predictor = @trackingPredictor;

% Initialize: get the first prediction
w = zeros(sum(colsInCell(Dtraining{1}.x)), 1);
for n = 1:length(Dtraining)
    d = Dtraining{n};
    d.y_dot = d.y_partial;
    d.y_hat = trackingPredictor(d, w, ...
                        settings.penalty_augment, settings.penalty_search_space);
    Dtraining(n) = {d};
end

ticID = tic;
[w, Dtraining, A, B, R, W, cccp_meta] = slpa_bmrm(w, Dtraining, functors, settings);
println('\nTotal training time: %g', toc(ticID));
```

5 Closing Remarks

5.1 Contact

Please forward any suggestions, bug reports, questions to `xinghua.lou@gmail.com`. Your feedback is highly appreciated.

5.2 Future Development

- Adding the cutting-plane training methods in [5].
- Support stochastic gradient descent (SGD) for large training data.
- Your ideas? Your contributions?

References

[1] X. Lou and F. A. Hamprecht. Structured Learning for Cell Tracking. In *Neural Information Processing Systems (NIPS)*, 2011.

[2] X. Lou and F. A. Hamprecht. Structured Learning from Partial Annotation. In *International Conference on Machine Learning (ICML)*, 2012.

[3] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Meeting of the Association for Computational Linguistics (ACL)*, 2002.

[4] E. Fernandes and U. Brefeld. Learning from partially annotated sequences. In *European Conference on Machine Learning (ECML)*, 2011.

[5] T. Joachims, T. Finley, and Chun-Nam Yu. Cutting-Plane Training of Structural SVMs. *Machine Learning*, 77(1):27–59, 2009.

[6] B. Taskar, C. Guestrin, and D. Koller. Max-Margin Markov Networks. In *Neural Information Processing Systems (NIPS)*, 2003.

[7] C. H. Teo, S. V. N. Vishwanthan, A. J. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 11:311–365, 2010.

[8] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large Margin Methods for Structured and Interdependent Output Variables. *Journal of Machine Learning Research*, 6(2):1453, 2006.