Regularizing Structured Classifier with Conditional Probabilistic Constraints for Semi-supervised Learning

Vincent W. Zheng  
Advanced Digital Sciences Center, Singapore  
vincent.zheng@adsc.com.sg

Kevin Chen-Chuan Chang  
University of Illinois at Urbana Champaign, USA  
kcchang@illinois.edu

ABSTRACT

Constraints have been shown as an effective way to incorporate unlabeled data for semi-supervised structured classification. We recognize that, constraints are often conditional and probabilistic; moreover, a constraint can have its condition depend on either just observations (which we call x-type constraint) or even hidden variables (which we call y-type constraint). We wish to design a constraint formulation that can flexibly model the constraint probability for both x-type and y-type constraints, and later use it to regularize general structured classifiers for semi-supervision. Surprisingly, none of the existing models have such a constraint formulation. Thus in this paper, we propose a new conditional probabilistic formulation for modeling both x-type and y-type constraints. We also recognize the inference complication for y-type constraint, and propose a systematic selective evaluation approach to efficiently realize the constraints. Finally, we evaluate our model in three applications, including named entity recognition, part-of-speech tagging and entity information extraction, with totally nine data sets. We show that our model is generally more accurate and efficient than the state-of-the-art baselines. Our code and data are available at https://bitbucket.org/vwz/cikm2016-cpf/.

KEYWORDS

Structured Classifier, Conditional Probabilistic Constraint

1. INTRODUCTION

Structured classification has many applications, including Named Entity Recognition (NER) [10], Part-Of-Speech (POS) tagging [20], Entity Information Extraction (EIE) [4] and so on [19, 30]. In practice, many structured classification tasks have limited labeled data. There is a recent trend of using constraints as “weak supervision” to leverage unlabeled data for semi-supervised structured classification [5, 11, 25]. Standard supervision relies on labeled examples to guide training, whereas weak supervision relies on labeled features to guide predictions. For example, in NER for citation extraction, a useful feature is a token being a four-digit string of either “20xx” or “19xx”. This feature is labeled as Date [8], meaning that: if a token is “20xx” or “19xx”, then its label is Date. By constraining a model’s predictions on the unlabeled data to be consistent with this labeled feature, we achieve semi-supervised training.

Conditional probabilistic constraint. Firstly, constraint is often conditional. A constraint is commonly expressed as an if-then statement, corresponding to “implication” in first-order logic. In addition to the Date constraint, another useful NER constraint is: if two fields are both Title, then there is no punctuation between them. Informally, a constraint’s condition is defined by the if-part of its statement. Moreover, a constraint condition can depend on: either only observations (e.g., whether a token is “20xx” or “19xx” in the Date constraint), or even labels (e.g., whether two fields’ labels are Title in the other constraint). Secondly, constraint is often probabilistic too. For example, a token “1900” sometimes is a volume number instead of Date. That is, in some examples, a constraint statement’s then-part is false, given its if-part as true.

Conditional probabilistic constraints are common and useful, but not easy to model. Let us illustrate via a toy example in Figure 1(a).

Toy example. In Figure 1(a), we consider the simplest structured classification task of binary sequence classification. We have a labeled sequence \( D_L = (X_1, Y_1) = \{(x_{1,k}, y_{1,k})|k = 1, \ldots, 5\} \): first four nodes \( x_{1,1}, \ldots, x_{1,4} \) are Positive (colored red, \( y_{1,k} = +1 \) for \( k = 1, \ldots, 4 \)) and the fifth node \( x_{1,5} \) is Negative (colored green, \( y_{1,5} = -1 \)). We also have three unlabeled training sequences \( D_U = \{X_2, X_3, X_4\} = \{x_{i,k}|i = 2, 3, 4, k = 1, \ldots, 4\} \). The ground truth of \( D_U \) is: nodes on the left hand side of the classification boundary (colored blue) are Positive, and those on the right are Negative. For each node, we extract features based on row index (1 to 4), column index (1 to 5) and shape (circle, square, triangle); e.g., the square node in the fourth row, third column \( x_{3,4} \) has a feature vector of \([0, 0, 0, 1, \text{row}, 0, 0, 1, 0, \text{column}, 0, 1, 0, \text{shape}]\).

Due to limited labeled data, training a Conditional Random Field (CRF) [15] with \( D_L \) leads to significant error. As shown in Figure 2(a), CRF predicts all the \( D_U \) nodes\(^1\) in columns 1 to 4 as Positive. This is because \( Y_1 \) is row-independent, the resulting CRF will classify data only based on columns and shapes.

Different kinds of conditional probabilistic constraints can help, e.g., \( TC_1 \) and \( TC_2 \) in Table 1. Firstly, x-type constraint \( TC_1 \) holds with a 50% probability. \( TC_1 \)’s if-part only depends on observation (i.e., whether a node \( x \) is in column 1 to 4). We call a constraint whose condition is purely based on observation \( x \)’s as an x-type constraint. \( TC_1 \) holds with a 50% probability, as only half of the unlabeled nodes in columns 1 to 4 are truly Positive. Secondly, y-type constraint \( TC_2 \) holds with a 67% probability. \( TC_2 \)’s if-part also depends on labels (i.e., whether two nodes have their labels \( y = +1 \) and \( y' = +1 \)). We call a constraint whose condition is also

\(^1\)We only use \( D_U \) as test data in this toy example for easier illustration. In general, we have a separate test data set.
If a node is in column 1 to 4, then it is y-type constraint. More discussions are in Section 2.

Constraint formulation

- If two nodes are both Positive, then they have the same shape.
- If a node is in column 1 to 4, then it is Positive.

Table 1: Constraints for Figure 1(a).

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Constraint statement</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC1</td>
<td>x-type</td>
<td>If a node is in column 1 to 4, then it is Positive</td>
<td>50%</td>
</tr>
<tr>
<td>TC2</td>
<td>y-type</td>
<td>If two nodes are both Positive, then they have the same shape</td>
<td>67%</td>
</tr>
</tbody>
</table>

Challenge 1. Constraint formulation. We desire a constraint formulation that can flexibly model the constraint probability for both x-type and y-type constraints. The reasons are: 1) probability is the most direct way to quantify the constraint uncertainty; 2) both x-type and y-type constraints are natural forms of domain knowledge, thus we should support both of them. Besides, we also want to use this constraint formulation to regularize general structured classifiers (like CRF etc.) for semi-supervision ultimately. None of the existing work has such a formulation. Probabilistic logic models, such as Markov Logic Network [24] and Probabilistic Soft Logic (PSL) [3], appear as a natural choice. But firstly, their classifiers are logic-based, whereas we prefer using constraints in non-logic structured classifiers that are popular in many applications [19, 20, 30]. Besides, the probabilistic logic models actually use the first-order logic formulas as “features”, and only apply them to labeled data (for groundings) in training. Comparatively, we wish to model constraints as “labeled features”, which can be applied to unlabeled data for semi-supervision. On the other hand, the state-of-the-art non-logic constraint models [8, 10, 20, 21] also cannot meet our desired formulation, because: 1) they do not have an explicit notion of “condition”, and so far only implicitly consider the x-type constraint conditions in their formulations; 2) their using one single feature function to formulate a constraint is unable to model the probability of y-type constraint. More discussions are in Section 2.

Challenge 2. Inference complication. Due to the hidden variable y’s in the condition, y-type constraints can lead to complication in inference. For example, a brute-force evaluation of TC2 checks the labels of every two nodes. This essentially creates a dependency between every two nodes, thus resulting in a complete graph over the unlabeled data as shown in Figure 1(b). Inference with a complete graph is expensive [14], as also shown empirically later in Table 4. When the nodes become more or the edges become higher order (e.g., some y-type constraint may involve more than two hidden variables at a time, thus resulting in a hypergraph), the inference is even more time consuming. Comparatively, x-type constraint is less likely to suffer from inference complication, as its condition on only x’s often limits the number of dependencies.

Our proposal. Our insight for constraint formulation is to explicitly model: 1) constraint condition, such that we can characterize x-type and y-type constraints; 2) constraint probability, such that we can quantify the uncertainty. In all, we propose a new Conditional Probabilistic Formulation (CPF). In CPF, we model the constraint condition with a Boolean function $g(x, y)$, where x is an “instance” (e.g., in TC2 it means two nodes) and y is the labels. $g(x, y)$ naturally expresses a y-type constraint condition as it involves y. Its reduced version $g(x)$ expresses an x-type constraint condition. To model the constraint probability, we first introduce a Boolean function $f(x, y)$ to denote constraint satisfiability. We let $f(x, y)$ return one if the constraint is satisfied on an instance x with its labels y, or zero otherwise. We then use both $f$ and $g$ to define the constraint probability by $P(f(x, y) = 1 | g(x, y) = 1)$, i.e., the probability of a constraint being satisfied when its condition is true.

Our insight for efficient inference is to do selective evaluation, such that we can focus on evaluating only those truly “relevant” instances (e.g., the nodes with both y’s as +1 in TC2). The rationale is that, the complication of using y-type constraint TC2 stems from the brute-force evaluation with every pair of hidden variables; however, since the constraint is conditional, we only care about those pairs that are truly “relevant” – if we can guess which two y’s are +1 in TC2, then we can save a lot of effort by only evaluating them. In all, we propose a novel selective evaluation approach, which first uses the labeled data to estimate the relevance of an instance, and then only evaluates the small set of relevant instances. Our selective evaluation differs from other efficient inference approaches on densely connected graphs [7, 12, 14], as it is tailored for constraints and can automatically identify which dependencies to evaluate.

We summarize our contributions as follows:

- To the best of our knowledge, we are the first to support conditional probabilistic constraints (both x-type and y-type) for semi-supervised structured classification.
- We propose a new CPF formulation to model the conditional probabilistic constraints and a systematic selective evaluation approach to ensure the efficient realization.
- We evaluate our model with three applications. We are both more accurate (the relative F1 improvement is 3.3%-15.0% in NER, 4.6%-15.9% in POS and 8.6%-24.2% in EIE)
efficient (the speedup is 3.5× in NER citation extraction, 0.7×-3.0× in POS, 4.5×-24.1× in EIE) than baselines.

2. RELATED WORK

Different constraint formulations. First of all, we distinguish our CPF from the probabilistic logic models, such as MLN [24] and PSL [3]. We are only similar to them in that, the domain knowledge used to specify constraints can be read as first-order logic (FOL). However, we neither formulate domain knowledge by FOL formulas, nor design logic-based Probabilistic Graphical Models (PGMs). Instead, we formulate labeled features, which can be flexibly incorporated into general PGMs (e.g., CRF). Moreover, in MLN and PSL, the FOL formulas are used as “features”, and only applied to labeled data (as the formulas need to be grounded) in training. Comparatively, our constraints are used as “labeled features” and applied to unlabeled data in training for semi-supervision. Finally, we are also different from the work using FOL in non-logic PGMs, because: 1) we have different goals, e.g., [26] uses FOL in matrix factorization, [1, 22] use FOL in unsupervised topic modeling, but we aim at semi-supervised structured classification; 2) we model constraint condition and probability, but none of them do.

Secondly, we are also different from other non-logic constraint models\(^2\), such as Gibbs Sampling with Non-local Information (GSNI) [2, 10], Constraint-Driven Learning (CODL) [8], Generalized Expectation (GE) [20] and Posterior Regularization (PR) [11, 13, 31, 21]. These existing models lack an explicit notion of “condition”, and they only implicitly consider x-type constraint condition in their formulations. PR often misses condition since it applies a constraint to every instance. GE, GSNI and CODL all implicitly consider x-type constraint condition, as they always limit their constraint evaluation on some specific instances x’s. These existing models do not explicitly quantify the constraint uncertainty by probability either. GE tries to match the marginal distribution of a feature to some empirical distribution, and it does not discuss the uncertainty for this matching. GSNI and CODL allow constraint uncertainty with different constraint weights, while PR models the slack in each constraint inequality. Such weights and slack do not directly quantify the constraint uncertainty, thus sometimes hard to interpret or specify them. Finally, these existing models all use one single feature function to formulate a constraint. GE’s constraint function measures the label marginal distribution of a feature; whereas PR/GSNI/CODL’s all measure some sort of constraint satisfiability. Unfortunately, it is not possible to model the constraint probability with one single feature function for y-type constraints. In contrast, we introduce two constraint feature functions, namely constraint condition function and constraint satisfiability function, to define the constraint probability.

We summarize different formulations for constraint in Table 2.

<table>
<thead>
<tr>
<th>Formulation</th>
<th>TrainData</th>
<th>Classifier</th>
<th>Cond</th>
<th>Uncertain</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLN</td>
<td>Boolean logic labeled</td>
<td>logic PGM logic weight</td>
<td>n.a.</td>
<td></td>
</tr>
<tr>
<td>PSL</td>
<td>Lukasiewicz logic labeled</td>
<td>logic PGM logic weight</td>
<td>n.a.</td>
<td></td>
</tr>
<tr>
<td>GSNI</td>
<td>labeled feature</td>
<td>n.a.</td>
<td>PGMS</td>
<td>x-type weight</td>
</tr>
<tr>
<td>CODL</td>
<td>labeled feature unlabeled</td>
<td>PGMS</td>
<td>x-type weight</td>
<td></td>
</tr>
<tr>
<td>PR</td>
<td>labeled feature unlabeled</td>
<td>PGMS</td>
<td>x-type slack</td>
<td></td>
</tr>
<tr>
<td>GE</td>
<td>labeled feature unlabeled</td>
<td>PGMS</td>
<td>x-type n.a.</td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>labeled feature unlabeled</td>
<td>PGMS</td>
<td>x-type, y-type probability</td>
<td></td>
</tr>
</tbody>
</table>

 Different settings of efficient inference. As y-type constraint condition involves hidden variables, we often face densely connected graphs in inference. Typical inference methods, such as pseudo-likelihood, loopy belief propagation and Markov Chain Monte Carlo, are not tailored for densely connected graph [27].

The recent efficient inference approaches for densely connected graphs are common in deliberately reducing the edge dependencies from the graph, e.g., by using a truncated Gaussian kernel [14, 7] or decomposing a full graph to trees [12]. Our selective evaluation has different applications and techniques from these approaches. First, selective evaluation is tailored for modeling constraints, while the above methods are for different applications, making their insights of reducing dependencies (e.g., the insight of [14, 7] is limited to use Gaussian filtering on images) not applicable to our problem. Second, selective evaluation explicitly learns which dependencies to reduce, while the above methods all rely on heuristics (i.e., kernel truncation in [14, 7] and structure decomposition in [12]) to reduce the dependencies. Finally, although selective evaluation also tries to simplify the graph, it is different from (sparse) structure learning [16, 32], because: 1) in structure learning, the graph is often fixed (e.g., [16] considers different images with the same amount of pixels), but selective evaluation can work with dynamic graphs; 2) structure learning is for feature selection, where once a feature is removed, all of its related edges are removed. Selective evaluation can remove a subset of edges from a particular feature.

3. CONSTRAINT MODELING

In this section, we formulate conditional probabilistic constraints, and incorporate them in semi-supervised structured classification.

3.1 Conditional Probabilistic Formulation

Constraint condition. We explicitly model constraint condition, so as to formally define x-type and y-type constraints. For an x-type constraint like TC\(_i\), the existing constraint models such as GE have used a Boolean indicator function \(\mathbf{1}(x)\) to check the constraint condition over a node \(x\). For a y-type constraint like TC\(_j\), we can similarly introduce a Boolean function to check the constraint condition, but we have to define it over hidden variables. Denote \(Y_U = \{y_{k_1}, k_2\} \mid k_1 = 2,3,4, k_2 = 1,...,5\) as the hidden variables for \(X_U\). Then we define a Boolean constraint condition function \(g(y) = 1\) if the two-dimensional hidden variable vector \(y = [y, y'] \in Y_U^2\) has both \(y = +1\) and \(y' = +1\), or \(g(y) = 0\) otherwise. Formally, we define

**Definition 1 (Constraint Condition).** A constraint condition function \(g : X_U^{d_1} \times Y_U^{d_2} \rightarrow \{0,1\}\), where \(d_1, d_2 \in \mathbb{N} \cup \{0\}\). \(g\) returns one if the constraint is applicable to an instance \(x\) and its labels \(y\), or zero otherwise.

An x-type constraint is a constraint having its constraint condition \(g\) with \(d_1 \in \mathbb{N}, d_2 = 0\); whereas a y-type constraint is a constraint having its condition \(g\) with \(d_1 \in \mathbb{N} \cup \{0\}, d_2 \in \mathbb{N}\). Generally, we consider \(d_1 = d_2\) if \(d_1 \neq 0\), as it is common in practice. We disallow \(d_1 = d_2 = 0\), as it degenerates the definition of \(g\).

Constraint probability. We explicitly quantify the constraint uncertainty, by estimating the probability that a constraint is satisfied given its condition is true. For the y-type constraint TC\(_j\), we first introduce a Boolean function to evaluate constraint satisfiability: we let \(f(x, y) = 1\) if two different nodes \(x = [x, x'] \in X_U^2\)
have the same shape and their labels $y = [y, y'] = [1, +1]$; or $f(x, y) = 0$ otherwise. Note that we define $f(x, y)$ to include $y$, so that we can focus on evaluating the constraint satisfiability within its application scope (i.e., when its condition is true). As a result, if $f(x, y) = 1$, then $TC_2$’s condition function $g(y) = 1$, but not vice versa. Formally, we define

**Definition 2** (Constraint Satisfiability). A constraint satisfaction function $f$ is a binary feature function $f : \mathbb{X}^d \times \mathbb{Y}^d \rightarrow \{0, 1\}$, $d \in \mathbb{N}$, which returns one if a constraint is satisfied on the instance $x$ and its labels $y$, or zero otherwise.

Thus the probability of $TC_2$ being satisfied when its condition is true is $\Pr(f(x, y) = 1 | g(y) = 1)$. For more general constraint conditions, we define the constraint probability as $P(f(x, y) = 1 | g(y) = 1)$. Similar to GE, we expect this conditional probability to match an empirical probability value, which we denote as $\eta \in [0, 1]$. In all, we propose a conditional probabilistic formulation (CPF) to unify x-type and y-type constraints.

**Definition 3** (CPF). A constraint $c$ is expressed as a triple $(g(x, y), f(x, y), \eta)$ with $P(f(x, y) = 1 | g(y) = 1) = \eta$, where $\eta \in [0, 1]$ is the empirical conditional probability of $c$.

In CPF we consider equality with $\eta$ (GE similarly considers equality in its formulation), but it is straightforward to extend to inequality like PR does (i.e., solve the inequality constrained optimization with Lagrangian [11]). We leave this extension for future work.

### 3.2 Learning with Constraints

After knowing how to express constraints, we now define the problem input/output for semi-supervised structured classification, and then develop the training objective function.

**Problem input/output.** We generalize the toy example in Figure 1(a) to define the problem input/output. Denote a set of labeled training data as $D_L = \{(X_i^{(1)}, Y_i^{(1)}) | i = 1, ..., n_L\}$, with each $(X_i^{(1)}, Y_i^{(1)}) = \{x_i^{(1)}, y_i^{(1)}\} | k = 1, ..., n_i\)$. Each $(X_i^{(1)}, Y_i^{(1)})$ is a labeled sentence in NER/POS or a Web page with multiple labeled text snippets in EIE. Each $x_i^{(1)} \in X$ is a token in NER/POS or a text snippet in EIE; $y_i^{(1)} \in Y$ is the token’s label or snippet’s label respectively. Similarly, denote a set of unlabeled training data as $D_U = \{(X_i^{(u)}, Y_i^{(u)}) | i = 1, ..., n_U\}$, with each $(X_i^{(u)}, Y_i^{(u)}) = \{x_i^{(u)} | k = 1, ..., n_i\}$. Denote a set of CPF constraints as $C = \{c_1, ..., c_n\}$. In training, we aim to output a multi-class classifier $g(y | X; C)$ trained from $D_L$ and $D_U$ and $C$, where $X$ is a set of observations. Denote a set of test data as $D_T = \{(X_t^{(i)}, Y_t^{(i)}) | i = 1, ..., n_T\}$, with each $(X_t^{(i)}, Y_t^{(i)}) = \{x_i^{(i)}, y_i^{(i)} | k = 1, ..., n_i\}$. In testing, we classify each $X_t^{(i)} \in D_T$ by $Y_i^{(i)} \rightarrow \text{arg max } q(Y | X_t^{(i)}; C)$, and compare with the corresponding ground truth $Y_t^{(i)}$ for evaluation.

**Modeling labeled data.** In structured classification, data are often interdependent; e.g., in Figure 1(a), the labeled nodes are connected as a sequence. To model such interdependencies, we use CRF [15]: in Figure 1(a), we denote the labeled nodes as $X = \{x_{1,k} | k = 1, ..., 5\}$, $Y = \{y_{1,k} | k = 1, ..., 5\}$, then we can model: 1) the dependency between a node $x_{1,k}$ to its hidden variable $y_{1,k}$ with a feature $h_1(x_{1,k}, y_{1,k}) \in \mathbb{R}$; 2) the dependency between two hidden variables $y_{1,k}$ and $y_{1,k+1}$ with a feature $h_2(y_{1,k}, y_{1,k+1}) \in \mathbb{R}$. For each $h_i$, we can sum up all the feature values over $x$. For example, let $h_1(x_{1,k}, y_{1,k})$, by stacking the $h_1(x, y)$’s into a vector $\mathbf{h}(X, Y)$, CRF tries to find $\theta$ that maximizes

$$P_\theta(Y | X) = \frac{1}{Z_\theta(X)} \exp\{\theta \cdot \mathbf{h}(X, Y)\},$$

where $Z_\theta(X) = \sum_y \exp\{\theta \cdot \mathbf{h}(X, Y)\}$ is a normalization term. In general, given $(X_L, Y_L)$, we can train the CRF model $\theta$ by optimizing the negative log-likelihood:

$$\mathcal{L}_\theta = -\frac{1}{n_L} \sum_{i=1}^{n_L} \log P_\theta(Y_i^{(1)} | X_i^{(1)}) + \frac{\gamma}{2} \|\theta\|_2^2,$$

where $\|\cdot\|_2$ is a $L_2$-norm and $\gamma \geq 0$ is a regularization parameter.

**Modeling unlabeled data by constraints.** We use constraints for semi-supervised learning, by firstly estimating the conditional probability $P(f(x, y) = 1 | g(y) = 1)$ on the unlabeled data with some target distribution, and then enforcing it to match $\eta$. Take $TC_2$ as an example: denote $g(Y_i) = \sum_{y \in Y_i}\delta g(y)$ by summing up $g$ over all the hidden variable pairs; similarly $f(X_U, Y_U) = \sum_{x \in X_U, y \in Y_i}\delta f(x, y)$. For a target distribution $q$ (whose form can be defined later), we denote $E_q[f(X_U, Y_U)]$ as the expected number of Positive node pairs of the same shape; $E_q[g(Y_i)]$ is the expected number of Positive node pairs. Then, we can empirically estimate $P(f(x, y) = 1 | g(y) = 1)$ by $E_q[f(X_U, Y_U)] / E_q[g(Y_i)]$.

We now generalize the derivation of the above $TC_2$ example. Denote $g(Y_i) = \sum_{x \in X_i, y \in Y_i}\delta g(x, y)$ and $f(X, Y_i) = \sum_{f \in \{f_1(X, Y_i), ..., f_{nc}(X, Y_i)\}}$ as summations over the instances $x_i \in X$ and their hidden variables $y_i \in Y_i$. In CPF, we estimate $P(f(x, y) = 1 | g(y) = 1)$ by $E_q\{g(Y_i)\} / E_q\{f(X, Y_i)\}$, and make it equal to $\eta$.

$$E_q[f(X, Y)] = \eta \cdot E_q[g(Y_i)]$$

(3)

For notation simplicity, let $\zeta(X, Y) = f(X, Y) - \eta \cdot g(X, Y)$.

Then $E_q[\zeta(X, Y)] = E_q[f(X, Y)] - \eta \cdot E_q[g(X, Y)] = 0$.

**Overall objective function.** We now combine the labeled data loss in Eq.(2) and the constraints on unlabeled data in Eq.(4) into one objective function. One question remains: what should be the form of the target distribution $q$ in Eq.(4)? Earlier GE work let $q = P_\theta(Y | X)$. But the recent study [5] suggests that, treating $q$ as a separate distribution to optimize rather than setting $q = P_\theta(Y | X)$ is easier for inference. We advocate this idea and optimize $q(Y | X; C)$ to: 1) meet the constraint in Eq.(4); 2) explain the labeled data, by making $q$ close to $P_\theta$ by KL divergence:

$$\min_{\theta, q \in \Delta} \mathcal{L}_\theta + \frac{\alpha_1}{n_L} \sum_{i=1}^{n_L} KL(q(Y | X_i^{(i)}; C) || P_\theta(Y | X_i^{(i)}))$$

s.t. $E_q[\zeta(X, Y)] = 0, \forall i$.

(5)

where $\Delta$ is a simplex s.t. $\sum_{Y} q(Y | X; C) = 1$. For optimization simplicity, we relax Eq.(5) to a regularized optimization:

$$\min_{\theta, q \in \Delta} \mathcal{L}_\theta + \frac{\alpha_1}{n_L} \sum_{i=1}^{n_L} KL(q(Y | X_i^{(i)}; C) || P_\theta(Y | X_i^{(i)})) + \frac{\alpha_2}{2n_U} \sum_{i=1}^{n_U} ||E_q[\zeta(X_i^{(i)}, Y_i^{(i)})]||_2^2.$$  

(6)

As we will discuss shortly, solving Eq.(6) is not trivial due to the complication of y-type constraints.

### 4. EFFICIENT REALIZATION

In this section we discuss the complication of y-type constraints, and propose a selective evaluation model to efficiently solve Eq.(6).


4.1 Inference Complication

The inference complication of y-type constraint is due to the constraint condition involving unknown hidden variables. Take TC2 as an example. Because we do not know which pair of nodes are Positive for the unlabeled data, we have to evaluate every pair of hidden variables, resulting in a complete (or at least densely connected) graph over the hidden variables, as shown in Figure 1(b).

To avoid inference with an over densely connected graph, the existing work often tries to carefully design the graph structure in advance. For example, previous constraint models have tried to carefully design a constraint, such that it only evaluates a limited number of hidden variables by conditioning on certain x’s [20] and a limited number of dependencies among the hidden variables [11]. Even in probabilistic logic, MLN [24] also has to assume that a meta-constraint, if its condition has two hidden variables, results in a complete (or at least densely connected) graph over the hidden variables by conditioning on certain x’s.

Specifically, we call a y-type constraint particularly lead to inference complication, we define constraint order. For example, we call a y-type constraint c as a first-order constraint, if its condition only has one hidden variable, i.e., g(x, y). We call c as a second-order constraint, if its condition has two hidden variables, i.e., g(x, y) where y ∈ Y^2. From the graph perspective, first-order constraint generates node-wise dependencies, second-order constraint generates edge-wise dependencies. Higher-order constraint is possible, but rarely used in practice. Both our CPF and selective evaluation are general to higher-order constraints.

4.2 Selective Evaluation

Before we derive the general solution for efficiently realizing y-type constraints in multi-class structured classification, let us use TC2 in the binary-class structured classification as an example to get some insight. For TC2’s inference complication, we ask the question: do we really have to evaluate all the pairs of hidden variables? The answer is no, because in fact we only care about those pairs that are truly Positive. For example, in Figure 1(c) if we can guess the nodes x_2, x_3, x_4, x_5, x_6, x_7, x_8, and x_9 as likely to be Positive, then we can focus on only evaluating the pairs from these nodes, which results in a much simpler graph for inference.

How can we guess which instances are truly relevant for a y-type constraint? For example, which node pair is truly relevant for TC2? Essentially, we need to estimate whether a node pair y = [y', y'] will have a large probability Pr(g(y) = 1). Formally, define a threshold as ε, then we call a node pair is likely to be relevant if Pr(g(y) = 1) ≥ ε. For TC2, we can associate an indicator δ(Pr(g(y) = 1) ≥ ε) with each node pair, so as to select which instances as truly relevant for evaluation. Thus, in Eq.(6), we can change each ζ(x, y) to ζ(x, y) · δ(Pr(g(y) = 1) ≥ ε). So far, we have two readily available distributions q(Y | X; C) and Pθ(Y | X) that can estimate Pr(g(y) = 1). If we use q to estimate Pr(g(y) = 1), then the expectation term E_q[ζ · δ] becomes a complicated form w.r.t. q, which is hard to optimize later. Hence, we choose Pθ to estimate Pr(g(y) = 1). In particular, for TC2 we let Pr(g(y) = 1) = Pθ(y = +1, y' = +1 | X).

In the following, we generalize our selective evaluation insight on TC2 to the multi-class structured classification setting. The key points are: 1) find the truly relevant instances for a y-type constraint; 2) use Pθ to estimate an instance’s relevance.

Objective function with selective evaluation. Generally, for a y-type constraint c, we estimate whether an instance x_i and its labels y_j are relevant to c by Pr(g_c(x_i, y_j) = 1). Denote y^* as the preferred labels for c, such that g_c(x_i, y_j) = y^* = 1. For example, the preferred labels for constraint CC2 in Table 3 are Title then. We can estimate Pr(g_c(x_i, y_j) = 1) = Pθ(y_j = y^*_j | X), or Pθ(y_j | X) for short. Denote ε_c as the selection threshold for c. For each ζ_c(x_i, y_j), we introduce a selection indicator δ(Pθ(y_j | X) ≥ ε_c).

For optimization convenience, we use log Pθ instead of Pθ in δ, thus we have δ(log Pθ(y_j | X) ≥ ε_c). This δ selects which instance to evaluate for c. Finally, we define ζ^*_c(x_i, y_j) as

ζ^*_c(x_i, y_j) = \begin{cases} \delta(\log Pθ(y_j | X) ≥ ε_c)ζ_c(x_i, y_j), & \text{if c is y-type;} \\ ζ_c(x_i, y_j), & \text{otherwise}. \end{cases} \tag{7}

By replacing the ζ in Eq.(6) with ζ^*_c = [ζ^*_1, ..., ζ^*_n_c], we have a new objective function with selective evaluation:

\[ \min_{δ, q ∈ Δ} L_θ + \frac{α_1}{n_U} \sum_{i=1}^{n_U} KL(q(Y_i | X_i; C) || Pθ(Y_i | X_i)) \]

\[ + \frac{α_2}{2n_U} \sum_{i=1}^{n_U} || E_q[ζ^*_c(X_i, Y_i)] ||^2. \tag{8} \]

Learning selection thresholds. In Eq.(7), different constraints can have different thresholds ε_c’s. This is because different constraints naturally have different amounts of relevant instances. However, setting thresholds is difficult. Let us again use TC2 as an example to draw some insight to see if we can automatically learn such thresholds. For TC2, a threshold value ε finds a set of Positive node pairs, by which we get a percentage about the relevant node pairs among all the node pairs. If the labeled data and the unlabeled data follow the same data distribution (which should be a mild assumption in practice), then the percentage estimated on the unlabeled data equals to that estimated on the labeled data.

We generalize the above relevant instance percentage estimation from binary-class in TC2 to multi-class with constraint preferred labels y^*. Formally, denote m_c as the number of instances for constraint c on unlabeled data (X, Y); i.e., m_c = |{(x_i, y_j)|x_i ∈ X^c, y_j ∈ Y^c}|. Denote π_c as the percentage of relevant instances over all the m_c instances. We can estimate π_c as

π_c = \frac{1}{m_c} \sum_{j=1}^{m_c} \delta(\log Pθ(y_j | X) ≥ ε_c)g(x_i, y_j).

Denote π_c ∈ [0, 1] as the empirical percentage of relevant instances over all instances. Then, we define a meta-constraint for constraint c as: π_c = P(π_c). For each unlabeled data sequence X_U ∈ D_U, we estimate one π_c. For multiple y-type constraints, we denote ε = [ε_1, ..., ε_n_c] and each ε_c ≤ 0 as it should be in the same scale with log Pθ(y_j | X). In all, we enforce the meta-constraints as an extra regularization term to Eq.(8):

\[ \min_{δ, q ∈ Δ} L_θ + \frac{α_1}{n_U} \sum_{i=1}^{n_U} KL(q(Y_i | X_i; C) || Pθ(Y_i | X_i)) \]

\[ + \frac{α_2}{2n_U} \sum_{i=1}^{n_U} || E_q[ζ^*_c(X_i, Y_i)] ||^2 \]

\[ + \frac{α_3}{2n_U} \sum_{c ∈ C_y} \sum_{i=1}^{n_U} || m_c(π_c | π_c - π_c) ||^2, \tag{9} \]

where α_3 ≥ 0 is a trade-off parameter. Eq.(9) is our final objective function with selective evaluation and automatic threshold learning.

5. LEARNING ALGORITHM

In this section, we discuss the model training to solve Eq.(9), the model testing, finally the overall algorithm and its complexity.
Algorithm 1 CPF Training and Testing Algorithm

Input: Labeled training data \( D_L \), unlabeled training data \( D_U \), (labeled) test data \( D_T \) and a set of CPF constraints \( C \).

Output: The labels for the test data \( D_T \).

Offline Training:
1. Initialize parameters \( \theta, \epsilon \) and \( \lambda \).
2. While not converged
   3. \( \text{Foreach } \{X^{(i)}_k, Y^{(i)}_k\} \in \text{RandomShuffle}(D_L) \) do
      4. Fix \( \lambda \) and \( \epsilon \), optimize \( \theta \);
      5. Fix \( \lambda \) and \( \theta \), optimize \( \epsilon \);
      6. \( \text{Foreach } X^{(i)}_k \in \text{RandomShuffle}(D_U) \) do
         7. Fix \( \theta \) and \( \epsilon \), optimize \( \lambda \);
      8. Fix \( \lambda \) and \( \theta \), optimize \( \epsilon \);

Online Testing:
9. \( \text{Foreach } X^{(i)}_k \in D_T \) do
10. For \( \text{iter} = 1 \rightarrow T_2 \) do
11. \( \text{For } x^{(i)}_j \in X^{(i)}_k \) do
12. Sample \( y^{(i)}_j \) by Eq.(11) with earlier predictions \( y^{(t-1)}_{j-1} \).

5.1 Model Learning

Training. To learn the parameters \( \theta, q \) and \( \epsilon \), we use alternating optimization; i.e., every \( \theta \) we optimize one parameter while fixing the others, then we iteratively minimize Eq.(9) w.r.t. each parameter until the objective function value in Eq.(9) is not further reducing or the reducing amount is below some threshold (i.e., the optimization converges). We leave the optimization details in the appendix. In summary, after optimization, we obtain all the \( \theta^* \), \( q^* \) and \( \epsilon^* \). Interestingly, \( q^* \) admits a parametric form after optimization:

\[
q^*(Y|X;C) = \frac{1}{Z_{\theta,\lambda}(X)} e^{\theta h(X,Y) + \lambda \zeta^*(X,Y)}, \tag{10}
\]

where \( Z_{\theta,\lambda}(X) = \sum_Y e^{\theta h(X,Y) + \lambda \zeta^*(X,Y)} \) is a normalization term. Based on Eq.(10), we have two remarks: first, each constraint feature \( \zeta^*_c(X,Y) \) has a weight \( \lambda_c \), which is learned automatically from the data. These \( \lambda_c \)'s can balance the contribution from the constraints and from the general features \( \theta \). Second, as \( \theta \) is a CRF model, each feature in \( \theta h(X,Y) \) creates a set of cliques in a factor graph defined over both \( X \) and \( Y \). Similarly, each constraint \( \zeta^*_c(X,Y) \) also creates a set of cliques over the graph. Therefore, \( q^*(Y|X;C) \) is also a CRF model, but it is defined over a more complex graph, whose cliques are characterized by both \( \theta \) and \( \zeta^* \). If either any feature in \( \theta \) or any constraint in \( \zeta^* \) is higher-order, then \( q^*(Y|X;C) \) becomes a higher-order CRF model.

Testing. In testing, we use \( q^*(Y|X;C) \) in Eq.(10) for prediction. We do not use \( P_\theta(Y|X) \) for prediction, because \( q^*(Y|X;C) \) considers both the features \( \theta \) and the constraints \( \zeta \) as shown in Eq.(10), while \( P_\theta(Y|X) \) only considers \( \theta \). Specifically, given one test entity's data \( X_T \), we use \( q^*(Y|X_T;C) \) to find a configuration

\[
Y^* \leftarrow \arg \min_Y -\log q^*(Y|X_T;C). \tag{11}
\]

We use Gibbs sampling [10] to do prediction.

5.2 Algorithm and Complexity

We summarize our CPF training and testing in Algorithm 1. In training, we employ stochastic gradient descent to update the parameters. Specifically, for each iteration, we first randomly shuffle all the labeled data and for each \( \{X^{(i)}_k, Y^{(i)}_k\} \) we do local updates of the model parameters (Steps 3-5). We do similar updates for the unlabeled data (Steps 6-8).

We analyze the computation complexity of Algorithm 1. Suppose each \( X^{(i)}_k \) produces a graph \( G_i \) of \( n_i \) hidden variable nodes (for \( y^{(i)}_k \)'s) and \( n_i \) observation nodes (for \( x^{(i)}_k \)'s). Suppose \( G_i \) has \( n'_i \) edges. Denote \( |G_i| = 2n_i + n'_i \) as the number of nodes and edges in \( G_i \). Thanks to selective evaluation, we often make \( G_i \) sparse, i.e., \( |G_i| \ll 2n_i + n'_i \). In training, optimizing \( \theta, \lambda \) and \( \epsilon \) needs Loopy Belief Propagation (LBP) inference to compute the gradients. Denote the number of iteration for LBP as \( T_3 \). Then, the inference complexity with one \( X^{(i)}_k \) is \( O(T_3 \times |G_i|) \). Denote the number of constraints as \( |C| \). In addition to inference, computing constraint gradients also takes \( O(|C| \times |G_i|) \). Then, the overall complexity of updating parameters in one training iteration is \( O(T_3 \times |G_i|) + \sum_i |D_L \times |D_U| \times |C| \). In testing, for each \( X^{(i)}_k \), we go through all its hidden variables one by one in one sampling iteration. For each hidden variable, we need to evaluate constraints. Denote the number of sampling iteration in testing as \( T_2 \). Thus the total complexity is \( O(T_2 \times |C| \times |G_i|) \).

6. EXPERIMENTS

As \( \gamma \)-type constraints have never been identified and supported before, we focus on validating: 1) expressiveness of our CPF to model \( \gamma \)-type constraints; 2) efficiency of our selective evaluation to realize \( \gamma \)-type constraints. We extensively evaluate our model with one toy example and three typical structured classification applications: NER, POS and EIE, with totally nine data sets.

6.1 Experimental Setup

Data sets. For NER, we use two public data sets, including a citation extraction data set\(^3\) as used in [8], and a seminar extraction data set\(^4\) as used in [10]. For the citation data set, we use the same training-testing data splits as [8]. For the seminar data set, we use 20 seminar emails as \( D_L \), 65% of the other seminar emails as \( D_U \), and the remaining emails as \( D_T \). In both data sets, we follow [8] and [10] to extract the features as \( h \) in Eq.(1).

For POS, we use four public data sets from the CoNLL-X Shared Task\(^5\), which has POS tags for four different languages: dutch, danish, portuguese and swedish. We use the same training-testing data splits as provided by [6]. For each language, we use 100 sentences as \( D_L \), and the remaining training data as unlabeled data. We also follow [17] to map the POS tags to 12 universal tags and extract the features as \( h \) in Eq.(1).

For EIE, we collected three data sets by our own from the following domains: researcher, car and product. For each domain, we prepared a list of entities (e.g., researchers). For each entity, we collected a set of Web pages (on average five to ten). We further parsed each page into a set of text snippets. In total, we have 48.2K snippets for researcher domain, 25.3K snippets for car domain and 4.9K snippets for product domain. EIE aims to predict a label (e.g., in researcher domain, labels are \( \text{Bio, Education, etc.} \)) for each text snippet. For the researcher data set, we have labeled\(^6\) the text snippets of 100 entities as \( D_L \), and the snippets of another 103 entities as \( D_T \). We leave all the other 800 entities as \( D_U \). For the car data set, we have 10 entities as \( D_L \), 18 entities as \( D_U \) and 115 entities as \( D_T \). For the product data set, we have only one entity as \( D_L \), 49 entities as \( D_U \) and 34 entities as \( D_T \).

\(^3\)http://cogcomp.cs.illinois.edu/Data/CODL_data/data.tgz
\(^4\)http://people.cs.umass.edu/~mccallum/data.html
\(^5\)http://iti.uvt.nl/conll/free_data.html
\(^6\)The labeling was done by two human judges, who achieved an agreement of 84% for researcher, 79% for car and 93% for product.
## Constraints

We follow two guidelines to design constraints for the experiments. First, as none of the existing work supports y-type constraints, we only focus on experimenting with y-type constraints. Second, we consider y-type constraints as up to second or higher order, and the admissible ADJ field is preceded by pronoun keywords for Dutch/Danish, or the NUM field matches number for Portuguese/Swedish.

### Baselines

We use alternating projection to implement GE, which gives the empirical results shown in Figure 1(a). In Figure 2, we showed the predictions by our CPF, as explained earlier in Section 1. CPF mistakenly predicts all of nodes in columns 1 to 4 as Negative due to limited labeled data. CPF can perfectly classify the data thanks to the constraint formulation and its efficient realization.

### Parameters

Our CPF has three parameters, $\alpha_1$, $\alpha_2$ and $\alpha_3$. Due to space limit, we will skip the figures about the performance change w.r.t. different $\alpha_i$-values. In summary, our empirical results show that all these $\alpha_i$'s can be easily tuned within the range of [0.01, 1] with relatively stable performance. For NER citation and seminar extraction, we both set $\alpha_1 = 0.01, \alpha_2 = 1$ and $\alpha_3 = 0.01$. For POS, we set $\alpha_1 = 0.01$; we also set $\alpha_2 = 1$ and $\alpha_3 = 0.01$ for Swedish, $\alpha_2 = 0.1$ and $\alpha_3 = 0.1$ for Danish, Dutch and Portuguese. For EIE, we set $\alpha_2 = 1, \alpha_3 = 0.1$; we also set $\alpha_1 = 0.01$ for researcher and car, $\alpha_1 = 1$ for product. Besides, the CRF parameter $\gamma$ is not our core parameter; we fix $\gamma = 0.001$ for NER and POS, and $\gamma = 0.1$ for EIE throughout our experiments.

### Environment

We run all the experiments on the machines equipped with eight 3.50GHz Intel Xeon(R) CPUs and 16GB memory.

## 6.2 Toy Example

### Expressiveness

We use alternating projection to implement GE, which gives the same objective as PR [5]. Thus we do not compare with PR again.
Efficiency. To validate the efficiency of selective evaluation, we consider a brute-force implementation of CPF, which denoted as CPF-. Specifically, CPF- is the solution for Eq.(6), where we re- alized the y-type constraints without selective evaluation. We re- ported the average training time by five runs in milliseconds in Ta- ble 4. As we can see, CPF can be 0.6 times faster than CPF-.

6.3 Named Entity Recognition

Expressiveness. We evaluate the usefulness of y-type constraints $CC_1, CC_2, CC_3$ for the citation data set and $SC_1, SC_2, SC_3$ for the seminar data set. In Figure 3(a), we add constraints one by one in each data set according to their constraint ID; e.g., in the citation data set, when the number of constraint is $K (K = 1, 2, 3)$, we use $CC_1, ..., CC_K$. We also use the same way of adding constraints in the POS and EIE data sets later. As shown in Figure 3(a), adding more constraints basically achieves higher performance.

We also compare with the baselines under varying labeled data sizes. As shown in Figure 4(a), generally our CPF is better than the baselines, since the baselines do not really support y-type constraints by nature. When the number of labeled data examples is 25, our CPF can achieve 4.9%-15.0% relative F1 improvement than the baselines in the citation data set, and 3.3%-6.9% relative F1 improvement in the seminar data set. Both improvements are significant, with the two-tailed t-test p-value less than 0.05.

Efficiency. We evaluate the efficiency by comparing CPF with GE, which shares a similar setting with us in using constraint with un- labeled data through expectation. The other baselines are not qual- ified for efficiency comparison. Specifically, CRF does not have constraints, GSNI does not use constraints in training, and CODL uses constraints in a bootstrapping framework. It is neither fair nor meaningful to compare with these baselines with different settings. As shown in Figure 5(a), as in GE we consider brute force implementation of y-type constraint, our CPF can be 3.5 × faster than GE in the citation data set, when using all the three constraints. It is interesting to see that, CPF does not gain any speed up in the semi- nar data set. The reason is that, both $SC_2$ and $SC_3$ are deliberately designed to have a limited application scope (e.g., conditioning on $x$’s being number fields). Since there are few number fields in the seminar emails, both $SC_2$ and $SC_3$ only create a small amount of interdependencies in extraction. When the graph is so simple, GE can do inference efficiently, and it can be even faster than CPF since it saves the computation for selective evaluation. However, as we shall see shortly in both POS and EIE, not all the y-type constraints can be such simple.

6.4 Part-Of-Speech Tagging

Expressiveness. We evaluate the usefulness of y-type constraints $OC_1, OC_2, OC_3$ for the four POS data sets. As shown in Figure 3(b), all the constraints help performance.

We also compare with the baselines under varying labeled data sizes. As shown in Figure 4(b), our CPF is mostly more accurate than the baselines, except in the danish data set (even in this case, CPF is only slightly worse than GE by 1.3%, but still better than other baselines by 5.7%-8.0%). When the number of labeled data examples is 100, our CPF is 6.4%-15.9% (dutch) / 5.4%-10.2% (portuguese) / 4.6%-11.4% (swedish) better than the baselines. All the improvements are significant, with two-tailed t-test $p \leq 0.10$.

Efficiency. As shown in Figure 5(b), when using all the three con- straints in each data set, our CPF is $0.7 \times 2.6 \times 3.0 \times 2.6 \times$ faster than GE on the dutch/danish/portuguese/swedish data sets. It is worth noting that, when the number of constraint is one (i.e., only using $OC_1$), CPF and GE tend to achieve similar efficiency. This is because, $OC_1$ is a first-order constraint, which only imposes node-wise dependencies but no edge-wise dependencies among the hidden variables. The resulting graph is thus easy for inference to both CPF and GE. We will again observe such comparable efficiency in EIE when using only first-order constraints.

6.5 Entity Information Extraction

Expressiveness. We evaluate the usefulness of y-type constraints $RC_1, RC_2, RC_3$ for the researcher data sets, $AC_1, AC_2, AC_3$ for the car data set and $PC_1, PC_2, PC_3$ for the product data set. As shown in Figure 6, all the constraints improve the performance.

We also compare CPF with the baselines as training data amount changes from 20% to 100% for each entity. Figure 7 shows CPF is generally better than the baselines. When using 100% data for each entity, CPF achieves 9.1%-24.2% (researcher) / 8.6%-17.3% (car) /
9.5%-17.1% (product) relative F1 improvement over the baselines. All the improvements are significant, with $t$-test $p \leq 0.05$.

**Efficiency.** As shown in Figure 8, when using all the three constraints in each data set, CPF is $19.0 \times 4.5 \times 24.1 \times$ faster than the baselines in the researcher /car/product data sets.

7. **CONCLUSIONS**

In this paper, we study how to formulate conditional probabilistic constraints and use them to regularize the general structured classifiers for semi-supervised learning. We first propose a new conditional probabilistic formulation to model both x-type and y-type constraints, where we explicitly characterize their constraint conditions and quantify their constraint probabilities. We then propose a systematic selective evaluation approach to address the inference complication issue raised by the y-type constraint’s involving unknown hidden variables in its condition. So far as we know, we are the first to support both x-type and y-type constraints for semi-supervised structured classification. We evaluate our model with three popular structured classification applications, including NER, POS and EIE. Our experiments show that we are both more accurate (the relative F1 improvement is 3.3%-15.0% in NER, 4.6%-15.9% in POS and 8.6%-24.2% in EIE) and efficient (the speedup is $3.5 \times$ in NER citation extraction, $0.7 \times 3.0 \times$ in POS, $4.5 \times 24.1 \times$ in EIE) than the state-of-the-art baselines.

In the future, we plan to add inequality in CPF (as remarked in Section 3.1). We also want to study how to best select constraints.

8. **ACKNOWLEDGMENTS**

This study is supported by the research grant for the Human-Centered Cyber-physical Systems Programme at the Advanced Digital Sciences Center from Singapore’s Agency for Science, Technology and Research (A*STAR). Vincent Zheng is also supported by the research grant (No. 61502418) from the National Nature Science Foundation of China.

9. **REFERENCES**


In this appendix, we present the details for our model learning.

**Fix \( \theta \) and \( \epsilon \), optimize \( q \).** By fixing \( \theta \) and \( \epsilon \)'s, we have \( C_0 \) in Eq.(9) constant. By re-organizing the constant \( \alpha_1 \), we optimize:

\[
\min_{q \in \Delta} l(q) = \frac{1}{n_U} \sum_{i=1}^{n_U} KL(q(Y_i|X_i))|P_0(Y_i|X_i)) + \frac{\alpha_2}{2\alpha_1 n_U} \sum_{i=1}^{n_U} ||E_q[\tilde{\psi}^2(X_i,Y_i)]||^2_2
\]  

(12)

Optimizing Eq.(12) in the primal form is hard, thus we consider its dual [9]. Simply put, by Fenchel duality, we have: \( \inf_{q} \{ \phi(u) + \psi(Au) \} = \sup_{\alpha, \lambda} \{ -\phi^*(\alpha^T \lambda) - \psi^*(-\lambda) \} \), where \( u \) is primal variable, \( \lambda \) is dual variable, \( \phi \) and \( \psi \) are the conjugates of \( \phi \) and \( \psi \). Our goal is to organize \( l(q) \) as \( \phi(q) \) and \( \psi(Aq) \); i.e., \( \psi(q) \) is the KL term, \( \phi(Aq) \) is the expectation norm term with each \( A_i(x) = \tilde{\psi}(X_i,Y_i) \). Then, we can use the definition of conjugate to compute each \( \phi^*(A_i \lambda) = \sup_{q \in \Delta} A_i \lambda : q(Y_i|X_i) - KL(q(Y_i|X_i)||P_0(Y_i|X_i)) \geq \log \sum_{\lambda} P_0(Y_i|X_i)e^{A_i \lambda} \), where step 2 holds iff \( q^*(Y_i|X_i) = \frac{P_0(Y_i|X_i)e^{A_i \lambda}}{\sum_{\lambda} P_0(Y_i|X_i)e^{A_i \lambda}} \). Plug in definitions of \( A_i \) and \( P_0 \), we have Eq.(10). Similarly, we can get \( \psi^*(-\lambda) = \frac{1}{\alpha_2} \sqrt{||\lambda||^2} \), thus giving the dual max \( \lambda^T \lambda \) = \( \frac{1}{\alpha_2} \sum_{i=1}^{n_U} [\log Z_0,\lambda(X_i) + \log Z_0(X_i)] - \frac{1}{\alpha_2} \sum_{i=1}^{n_U} ||\lambda||^2_2 \). The gradient of \( \lambda \) is \( \frac{\partial\lambda}{\partial \lambda} = \frac{1}{\alpha_2} \sum_{i=1}^{n_U} E_q[\tilde{\psi}(X_i,Y_i)] = \frac{1}{\alpha_2} \lambda \). We use Loopy Belief Propagation to compute this gradient.

**Fix \( q \) and \( \epsilon \), optimize \( \theta \).** By fixing \( q \) and \( \epsilon \)'s, we only have the y-type constraints contributing to \( \|E_q[\tilde{\psi}(X_i,Y_i)]\|_2^2 \). Denote the set of y-type constraints in use as \( C_0 \). Hence, we optimize:

\[
\min_{\theta} l(\theta) = L_\theta + \frac{\alpha_1}{2n_U} \sum_{i=1}^{n_U} KL(q(Y_i|X_i);C)||P_0(Y_i|X_i)) + \frac{\alpha_2}{2n_U} \sum_{i=1}^{n_U} \sum_{c \in C_0} \left[ E_q[\tilde{\psi}(X_i,Y_i)] \right]^2 + \frac{\alpha_3}{2n_U} \sum_{i=1}^{n_U} \sum_{c \in C_0} m_{c}(\pi_i(c) - \tilde{\pi}_c)^2 \right]
\]  

(13)

Its gradient is \( \frac{\partial l(\theta)}{\partial \theta} = \frac{\partial L_\theta}{\partial \theta} + \frac{\alpha_1}{2n_U} \sum_{i=1}^{n_U} \frac{\partial KL(q|P_0)}{\partial \theta} + \frac{\alpha_2}{2n_U} \sum_{i=1}^{n_U} \sum_{c \in C_0} \frac{\partial E_q[\tilde{\psi}(X_i,Y_i)]}{\partial \theta} + \frac{\alpha_3}{2n_U} \sum_{i=1}^{n_U} \sum_{c \in C_0} m_{c}(\pi_i(c) - \tilde{\pi}_c) \frac{\partial m_c}{\partial \theta}, \) where each partial derivative can be computed straightforwardly, thus details are skipped.

**Fix \( q \) and \( \theta \), optimize \( \epsilon \).** Re-organize the constant \( \alpha_2 \), and we have:

\[
\min_{\epsilon \leq 0} l(\epsilon) = \frac{1}{2n_U} \sum_{i=1}^{n_U} \sum_{c \in C_0} \left[ E_q[\tilde{\psi}(X_i,Y_i)] \right]^2 + \frac{\alpha_3}{2n_U} \sum_{i=1}^{n_U} \sum_{c \in C_0} m_{c}(\pi_i(c) - \tilde{\pi}_c)^2
\]  

(14)

As each \( \epsilon_i < 0 \), we define \( \epsilon_i = \log \sigma \) with \( \sigma = \frac{1}{1 + e^{\epsilon_i}} \in [0, 1] \), and the constrained optimization in Eq.(14) is casted into unconstrained optimization over \( l(\epsilon_i)'s \). Then the gradient becomes

\[
\frac{\partial l(\epsilon_i)}{\partial \epsilon_i} = \frac{1}{n_U} \sum_{i=1}^{n_U} E_q[\tilde{\psi}(X_i,Y_i)] \frac{\partial E_q[\tilde{\psi}(X_i,Y_i)]}{\partial \epsilon_i} + \frac{\alpha_3}{n_U} \sum_{i=1}^{n_U} m_{c}(\pi_i(c) - \tilde{\pi}_c) \frac{\partial m_c}{\partial \epsilon_i}, \]  

where each partial derivative can be computed straightforwardly, thus details are skipped.