University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 13: October 11

Lecturer: Alexander Schwing Scribe: Seo Taek Kong

1 Informal Overview

In this section, I attempt to describe the problem and methods to solve the problem that is understandable for those who have not had much practice with optimization and probability other than in this course. After this section, I plan to give a detailed explanation of what we went over in class for those who want a detailed understanding of the topic. Those who have a strong background in probability and/or optimization can skip to the next section.

Up to lecture 11, we discussed how to model a classification/regression model by deciding on the prediction rule (either a neural network or using scores $w^T x$, where w are the weights to be learned and x is the input). For lectures 12 and 13, we *assume* that we have access to some appropriately trained classification model and focus on how to improve classification by using the inherent *structure* of the problem, *without* modifying the models we have.

Consider the example described in class: we want to classify four pictures of four different handwritings of "Q", "U", "I", "Z". However, the person who wrote this word has sloppy handwriting and the "U" appears to be a "V". As English speakers, we know that "V" is unappropriate here and would understand this word as QUIZ. We would like to program a classifier that can also use this same line of thinking to decode the word as "QUIZ" and not "QVIZ". For this problem the *structure* can be thought of as belonging to a valid English word.

Suppose we have a model which, for each image, outputs the probability (score) the image is a letter, for all letters; if we input an image, the model outputs $\mathbb{P}(A), \mathbb{P}(B), \ldots, \mathbb{P}(Z)$. Independent prediction would just output the most likely letter for each image. Exhaustive search and dynamic programming as methods for structured prediction were discussed in the previous lecture and here we focus on defining the *program* and methods to solve it.

Exhaustive search obviously gives the best possible combination of the letters assuming we have good classifiers. We can deduce the program by thinking about exhaustive search. Suppose our trained classifiers give some score f for each letter and pair for each image and each pair of image. For example, the first image is "Q" with probability 0.9 and O with probability 0.1 giving $f_1(\text{"}Q\text{"}) = 0.5$, $f_1(\text{"}O\text{"}) = 0.1$ and others zero. The first two letters together are "QU" w.p. 0.7, "QV" w.p. 0.2, "OU" w.p. 0.05, and "OV" w.p. 0.05.

1. Since each image cannot simultaneously be two letters, so we want to specify in the program that if we predict the image as "Q", then we cannot predict it to be "O". This is why we have the first constraint, $\forall \mathbf{y}_r, b_r(\mathbf{y}_r) \in \{0,1\}$ where $b_{1,2}(\text{"}QU\text{"}) = 1$ means the first two images (as seen by the subscript) is predicted to be "QU".

2. In classification we are often concerned with likelihoods / probabilities, so we force our structured classifier to output distributions. Thus we get the constraints $b_r(\mathbf{y}_r) \geq 0 \ \forall r, \mathbf{y}_r$, $\sum_{\mathbf{y}_r} b_r(\mathbf{y}_r) = 1 \ \forall r$.

3. It wouldn't make sense to say the first two images are "QU" and "QV" with probability 0.7 and 0.2, but also say that the first image is "O" with probability 1. This is where we get the

"marginalization constraint":

$$b_{1,2}(\text{"}QU\text{"}) + b_{1,2}(\text{"}QV\text{"}) = b_1(\text{"}Q\text{"})$$
$$b_{1,2}(\text{"}QU\text{"}) + b_{1,2}(\text{"}OU\text{"}) = b_2(\text{"}U\text{"})$$
$$b_{1,2}(\text{"}OU\text{"}) + b_{1,2}(\text{"}OV\text{"}) = b_1(\text{"}O\text{"})$$
$$b_{1,2}(\text{"}QU\text{"}) + b_{1,2}(\text{"}OV\text{"}) = b_2(\text{"}V\text{"})$$

which can be written more generally as

$$\sum_{\mathbf{y}_p \setminus \mathbf{y}_r} b_p(\mathbf{y}_p) = b_r(\mathbf{y}_r).$$

Here we derived what constraints we should have for our decision rule. Observing each constraint closely, we can see they are linear (except the first one which is an integer constraint). We can also define a linear objective function as done in class. Thus, we came up with an integer linear program (ILP). Solving this directly is pretty much the same as using exhaustive search and may require K^D time.

Since we want to move away from exhaustive search for large problems, we need to work around solving this directly. The main difficulty in solving this problem is the integer constraints, so we just get rid of them with hopes that our solution to the program ends up being integer. Otherwise, we can either find an integer solution that is close to ours, or resolve using some other tricks.

After removing the integer constraints we have a linear program. There are many solvers that can find the optimal solution quickly, and if the solution ends up being integers we get the exact solution to the above ILP and have solved the program *exactly*. In this case, our prediction would be the corresponding $\arg\max_{\mathbf{y}_r} b_r(\mathbf{y}_r)$ which is the argument corresponding to $b_r(\mathbf{y}_r) = 1$.

An alternative to using an LP solver would be to solve the dual using convex optimization solvers. One advantage is that the dual has no constraints whereas the relaxed LP still has the marginalization constraint which can be challenging. The other constraints force each b_r to be a probability distribution for each r, and this is not too difficult to impose relative to the marginalization constraint. Thus we use Lagrange multipliers to *soften* the relaxed LP by adding the marginalization constraint with the Lagrange multipliers and find the dual. When we find a dual, we always obtain a relationship between the primal variables and the dual variables, i.e. $b_r = h(\lambda)$ where h is some function. Thus, solving the dual immediately gives the the solution to the primal variables. In this context, solving the dual is named *message-passing* because the dual variables λ can be thought of as messages giving information about the decision rule b_r.

Graph-cut solvers are used as another method to infer the best structured output. One limitation is that there must be some constraint on the *structure* of the output: in class we considered the case that outputs are binary-valued and the structures are considered only pairwise. When considering just this case, graph-cut solvers are efficient in inferring the optimal structured output. To use such solvers, we take a structured inference problem and convert it into a weighted graph by following an algorithm described near the last section. Graph-cut solvers can then compute the minimum cut or maximum flow (**not maximum cut, which is difficult to obtain**), giving the optimal structured output.

2 Introduction to Structured Prediction

2.1 Problem of Structured Prediction

We are interested in *structured prediction*, where we aim to predict an output $\mathbf{y} \in \mathcal{Y} = \{1, \ldots, K\}^D$ which may be correlated, or have *structure*, given an input $x \in \mathcal{X}$.

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \sum_{r \in \mathcal{R}} f_r(w, x, y_r) \tag{1}$$

We assume the score function $F(w, x, \hat{\mathbf{y}})$ can be decomposed as a sum of functions $\sum_{r \in \mathcal{R}} f_r(w, x, \mathbf{y}_r)$. In separate prediction, this is simply reduced to $\mathcal{R} = \{1, \ldots, D\}$, i.e. the outputs are independent. The resulting prediction is $\hat{\mathbf{y}} = \big(\hat{y}_1, \ldots, \hat{y}_D\big)$, where

$$\hat{y}_i = \arg\min_{y_i} f_i(w, x, y_i)$$

2.2 Motivation for Structured Prediction

Consider the following example which illustrates the difference between separate and structured prediction: A smart-phone user is either a cat-lover or a dog-lover, but never both. An engineer wants to include a prediction algorithm that can automatically label pictures in an album onto a user's smart-phone. Suppose the user has D pictures in his album with all pictures being either all dogs or all cats. We want to exploit this structure inherent in the output.

Example 1 (Separate Prediction) *For separate prediction, there are no restrictions on the structure, $\mathcal{R} = \{1, 2, \ldots, D\}$. Let the scores of each label be*

$$f_i(w, x, y_i) = \mathbb{P}\left(y_i = cat \,|x = image, w\right)$$

using a logistic regression model, i.e. $p(y_i = cat|x, w) = \frac{1}{1+e^{-w^T x}}$. with these scores we can predict the label of each picture in the album with the following decision rule for each i

$$\hat{y}_i = \begin{cases} cat \text{ if } f_i(w, x, y_i) \geq \frac{1}{2} \\ dog \text{ if } f_i(w, x, y_i) < \frac{1}{2} \end{cases}$$

Example 2 (Structured Prediction) *We can take advantage of the fact that the pictures the user has is either both dogs or both cats, and not a mixture by taking $\mathcal{R} = \{\{1, 2, \ldots, D\}\}$. To exploit the structure, suppose we use the following score function*

$$f_{\{1,2,\ldots,D\}}(w, x, (cat, cat, \ldots, cat)) = \sum_{i=1}^{D} \mathbb{P}\left(y_i = cat \,|x, w\right)$$

$$f_{\{1,2,\ldots,D\}}(w, x, (cat, cat, \ldots, dog)) = \sum_{i=1}^{D} \mathbb{P}\left(y_i = dog \,|x, w\right)$$

$$f_{\{1,2,\ldots,D\}}(w, x, (cat, cat, \ldots, cat)) = 0 \; \forall \boldsymbol{y} \notin \{(cat, cat, \ldots, cat), (dog, \ldots, dot)\}$$

Using the above scores, the prediction rule becomes

$$\hat{\boldsymbol{y}} = \big(\hat{y}_1, \ldots, \hat{y}_D\big) = \begin{cases} (cat, \ldots, cat) \text{ if } f_{\{1,\ldots,D\}}(w, x, cat, \ldots, cat) \geq f_{\{1,\ldots,D\}}(w, x, dog, \ldots, dog) \\ (dog, \ldots, dog) \;\; else \end{cases}$$

Now consider the following: We trained a logistic model so that the classification error given a picture is 0. However, each time the owner takes a picture, the image is blurred so much that a cat image becomes closer to a dog's image with probability $\epsilon \in (0, 1/2)$ and vice-versa. Assuming that a pet owner is a dog or cat lover with equal probability, the resulting error probability for the separate and structured prediction is given below:

$$P_e(\text{Separate Prediction}) = \mathbb{P}\left(\bigcup_{i=1}^{D}\{\hat{y}_i \neq y_i\}\right) = 1 - (1-\epsilon)^D \overset{D\to\infty}{\to} 1$$

and defining the random variable $X_k \sim P_X = Ber(1-\epsilon)$,

$$P_e(\text{Structured Prediction}) = \mathbb{P}\left(\sum_{k=1}^{D} X_k < \frac{D}{2}\right) = \sum_{k=0}^{\lfloor D/2 \rfloor} \binom{D}{k} (1-\epsilon)^k \epsilon^{n-k} \overset{D\to\infty}{\to} 0$$

From this example, it is clear that structured prediction gives an advantage over separate prediction.

2.3 Simple Algorithms for Structured Prediction

We discussed exhaustive search and dynamic programming methods for structured prediction, which have the following properties:

(a) Exhaustive Search is easy to implement and effective for many applications (when $K^D \sim 10^5$). When the dimension grows larger than this, we need more sophisticated methods to infer the output structure.

(b) Dynamic Programming (DP)) can significantly reduce the search space by breaking the problem into smaller sub-problems and solving only the necessary portions. However, it can only be applied to tree-structured outputs; a loop in the structure may yield infinite recursion.

3 Algorithms for Large, General Structured Outputs

3.1 Overview

Whenever the dimension of the output structure is large and has loops (i.e. not a tree-structured output), we must rely on some other method to infer the output. Integer Linear Programming (ILP) is a linear optimization problem (objective and constraints are linear / affine) when the feasible set is also constrained to require finite precision (i.e. it is a countable set). Generally these problems are NP-Complete; it is impossible to solve reasonably large problems.

As a resolution, we remove the integer constraints. This is called Linear Programming relaxation, since we are relaxing (or removing) the integer constraints. We can find the solution to the LP, and ff the optimal solution of the linear program consists of integers, then it is obviously also the solution to the ILP and we have solved the problem in polynomial time. When we are not so fortunate, we either have the lower bound (upper bound) for minimization (maximization) problems and we can use this as a guide to find the integer solution or an approximately optimal solution. Another method is to use *message passing* which is to solve the dual of the linear program relaxation. This is because dual problems are usually easier to solve, one reason being that the dual is an unconstrained optimization problem. After solving the dual, we can substitute the optimal dual variables into the relationship between the primal and dual variables, obtaining the solution to the LP.

3.2 Integer Linear Programming Formulation

The goal here is to formulate the inference task

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \sum_{r \in \mathcal{R}} f_r(w, x, \hat{\mathbf{y}}_r)$$

into a linear program. Take the example discussed in class, where we have $\mathcal{R} = \{1, 2\}^2$. Then the inference task reduces to finding the argument of

$$\max\{f_1(1) + f_2(1) + f_{1,2}(1,1), f_1(1) + f_2(2) + f_{1,2}(1,2), f_1(2) + f_2(1) + f_{1,2}(2,1), f_1(2) + f_2(2) + f_{1,2}(2,2)\}$$

where as in the notes the arguments w, x are dropped for simplicity, since they are the same for all scores. Since each argument is linear, the inference problem is equivalent to picking any of the above combinations. This can be quantitatively written as the following linear program with $\mathbf{b} = \big(b_1(1), b_1(2), b_2(1), b_2(2), b_{1,2}(1,1), \ldots, b_{1,2}(2,2)\big)^T$ and $\mathbf{f} = \big(f_1(1), f_1(2), f_2(1), \ldots, f_{1,2}(2,2)\big)$, where the subscripts of b denote the groups or structure we are interested, and the arguments of b_r are the values y_r can take:

$$\begin{aligned}
&\max_{\mathbf{b}} \mathbf{b}^T\mathbf{f} \\
&\text{S.T. } b_r(\mathbf{y}_r) \in \{0,1\}\ \forall r, \mathbf{y}_r && \text{(integer constraint)} \\
&\quad b_r(\mathbf{y}_r) \geq 0\ \forall r, \mathbf{y}_r && \text{(restating a part of the above constraint)} \\
&\quad \sum_{\mathbf{y}_r} b_r(\mathbf{y}_r) = 1\ \forall r && \text{(some } \mathbf{y}_r \text{ must be selected)} \\
&\quad \sum_{\mathbf{y}_p \setminus \mathbf{y}_r} b_p(\mathbf{y}_p) = b_r(\mathbf{y}_r)\ \forall r, p \in Parent(r), \mathbf{y}_r && \text{(marginalization to } \mathbf{y}_r\text{)}
\end{aligned}$$

where p being a parent of r means that r is a subset of p, e.g. $r = \{1\}, p = \{1, 2\}$.

3.3 Solving the ILP Approximately (Relaxed LP and Message Passing)

Again, solving an ILP is infeasible for large problems. Observe that the second and third constraints together force the vector $\big(b_1, b_2, b_{1,2}\big)$ to be a probability mass function. Denote the set of $\mathbf{b}$ satisfying the probability constraints and marginalization constraint as $\mathcal{C}$. Then a linear programming (LP) relaxation is given by removing the integer constraint:

$$\max_{\mathbf{b} \in \mathcal{C}} \mathbf{b}^T\mathbf{f} \tag{2}$$

One method to solve this is using any standard LP solver (interior point, ellipsoid, etc.) and (1) if the solution consists of integers, return solution or (2) return the upper bound (or lower bound for minimization problem) of the ILP if the solution is not integral, obtained by solving the corresponding relaxed LP.

Instead of directly solving this LP relaxation, we can use *Message Passing* which is just solving the dual of equation (2). First, the primal problem is put in standard form:

$$\begin{aligned}
&\min_{\mathbf{b}} \ -\sum_{r, \mathbf{y}_r} b_r(\mathbf{y}_r) f_r(\mathbf{y}_r) \\
&S.T. \ -b_r(\mathbf{y}_r) \leq 0 \ \ \forall r, \mathbf{y}_r \\
&\quad \sum_{\mathbf{y}_r} b_r(\mathbf{y}_r) - 1 = 0\ \forall r \\
&\quad \sum_{\mathbf{y}_p \setminus \mathbf{y}_r} b_p(\mathbf{y}_p) - b_r(\mathbf{y}_r) = 0\ \forall r, p, \mathbf{y}_r
\end{aligned}$$

The Lagrangian is constructed with Lagrange multipilers for the last constraint, with the domain of b_r being the probability simplex:

$$\mathcal{L}(\mathbf{b}, \lambda) = -\sum_{r,\mathbf{y}_r} b_r(\mathbf{y}_r) f_r(\mathbf{y}_r) + \sum_r \sum_{\mathbf{y}_r} \sum_{p \in P(r)} \lambda_{r \to p}(\mathbf{y}_r) \Big(\sum_{\mathbf{y}_p \setminus \mathbf{y}_r} \big(b_p(\mathbf{y}_p)\big) - b_r(\mathbf{y}_r) \Big)$$
$$= \sum_{r,\mathbf{y}_r} b_r(\mathbf{y}_r) \Big(- f_r(\mathbf{y}_r) - \sum_{p \in Parent(r)} \lambda_{r \to p}(\mathbf{y}_r) + \sum_{c \in C(r)} \lambda_{c \to r}(\mathbf{y}_c) \Big)$$

grouping over the primal variables $\mathbf{b}$ and because $\sum_{r,p \in P(r)} \lambda_{r \to p}(\mathbf{y}_r) \sum_{\mathbf{y}_p \setminus \mathbf{y}_r} b_p(\mathbf{y}_p) = \sum_{r,c \in C(r)} \lambda_{c \to r}(\mathbf{y}_c) b_r(\mathbf{y}_r)$.

The dual function is the Lagrange minimized with respect to the primal variables $\mathbf{b}$:

$$g(\lambda) = \min_{\mathbf{b} \in \Delta} \mathcal{L}(\mathbf{b}, \lambda)$$

where Δ is the probability simplex. In other words, find a probability distribution for each $r \in \mathcal{R}$ that minimizes the Lagrange; the solution is simply given by giving the most weight on that minimizing $\mathbf{y}_r$, i.e.

$$g(\lambda) = \sum_r \min_{\mathbf{y}_r} \Big(- f_r(\mathbf{y}_r) - \sum_{p \in P(r)} \lambda_{r \to p}(\mathbf{y}_r) + \sum_{c \in C(r)} \lambda_{c \to r}(\mathbf{y}_c) \Big)$$

By duality theory, $g(\lambda)$ is a convex function and the only constraint is that $\lambda \geq 0$ for each $r, p, \mathbf{y}_r$. **Once we find the λ^* maximizing $g(\lambda)$, we can substitute the resulting λ^* into $\mathcal{L}$ and find the minimizing b given** λ^*, i.e.

$$b^* = \arg\min_{\mathbf{b}} \mathcal{L}(\mathbf{b}, \lambda^*)$$
$$= \arg\min_{\mathbf{b}} \sum_{r,\mathbf{y}_r} b_r(\mathbf{y}_r) \Big(- f_r(\mathbf{y}_r) - \sum_{p \in P(r)} \lambda^*_{r \to p}(\mathbf{y}_r) + \sum_{c \in C(r)} \lambda^*_{c \to r}(\mathbf{y}_c) \Big)$$

which gives the solution to the relaxed linear program. Because this is a linear program, strong duality holds: $\max_\lambda g(\lambda) = \min_{\mathbf{b}} - \sum_{r,\mathbf{y}_r} b_r(\mathbf{y}_r) f_r(\mathbf{y}_r)$ subject to the relaxed constraints.

3.4 Graph Cut Solvers

Graph cut solvers can compute the minimum cut or equivalently maximum flow through a weighted graph. There is a general class of problems in which graph cut solvers can be used to solve structured inference [2], but in class we focus on the case that variables are binary-valued $y_d \in \{1, 2\}\ \forall d$. I will again focus on this case, explaining how to construct a graph given single $f_d(y_d)$ and pairwise scores $f_{i,j}(y_i, y_j)\ \forall i, j \in \{1, \ldots, D\}$ as introduced in [1]).

First, I will state a modified result shown in (Lemma 3.2, Theorem 4.1 in [1]).

Theorem 3 *Suppose the score function $F(w, x, \boldsymbol{y})$ can be decomposed into local score functions depending on one or two binary variables, i.e.*

$$F(w, x, \boldsymbol{y}) = \sum_{i \in \{1,\ldots,D\}} f_i(y_i) + \sum_{i<j} f_{i,j}(y_i, y_j)\ \forall \boldsymbol{y} \in \{1, 2\}^D.$$

Then, it is possible to use a min-cut / max-flow algorithm to find $\arg\max_{\boldsymbol{y}} F(w, x, \boldsymbol{y})$ *if all pairwise decompositions (i, j) satisfy the following inequality*

$$f_{i,j}(1, 1) + f_{i,j}(2, 2) \leq f_{i,j}(1, 2) + f_{i,j}(2, 1) \tag{3}$$

By this theorem, if we are capable of decomposing the score function into pairwise terms such that this inequality holds we can always construct a graph that can then be solved using graph-cut solvers to infer the structure we are trying to solve. Further, since this class of score function includes any grid-structured output (by the pairwise decomposition), graph-cut solvers can be used for a larger class of inference problems than can be solved using dynamic programming.

The sub-modularity condition (equation 3) implies that pair-wise weights of the graph constructed is positive. To construct the graph given the decomposition, label D vertices $\{1, \ldots, D\}$ in addition to source and terminal nodes as in Figure 1.

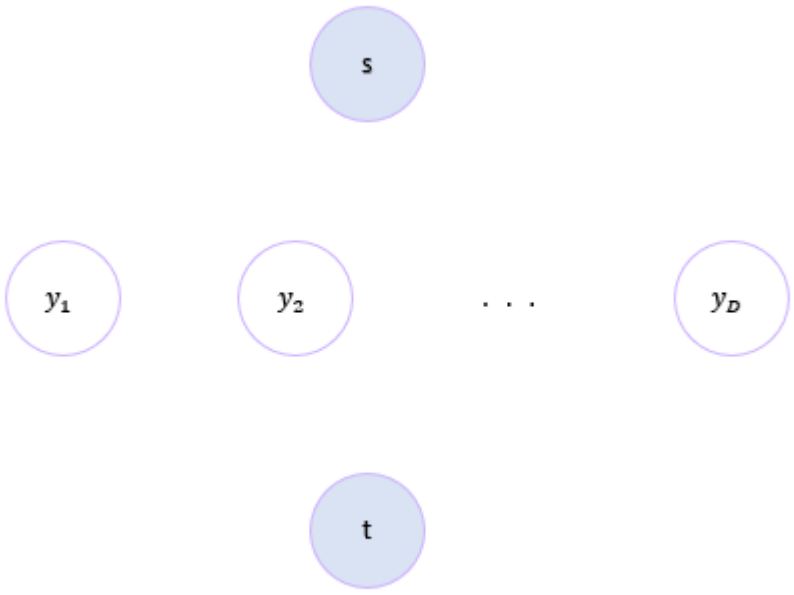

Figure 1: Listing nodes with $D = 3$

To add the edges / weights, consider the pairwise terms (in this case $f_{1,2}, f_{1,3}, f_{2,3}$), where each $f_{i,j}$ can be represented by a matrix

$$f_{i,j} = \begin{pmatrix} f_{i,j}(1,1) & f_{i,j}(1,2) \\ f_{i,j}(2,1) & f_{i,j}(2,2) \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

$$= \begin{pmatrix} A & A \\ A & A \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ C-A & C-A \end{pmatrix} + \begin{pmatrix} 0 & D-C \\ 0 & D-C \end{pmatrix} + \begin{pmatrix} 0 & B+C-A-D \\ 0 & 0 \end{pmatrix}$$

which is obtained by taking out the first element, pivoting the first row, the first column, and then adding the remaining terms. The last inequality follows because $f_{i,j}(y_i, y_j = 2) - f_{i,j}(y_i, y_j = 1)$ only depends on y_i and same for y_j.

For each pair, the graph will be constructed according to the following algorithm:

Algorithm

1. The second matrix in the last equation only depends on y_i whereas the third matrix depends only on y_j (e.g. looking at the second matrix, if $y_i = 1$ then $f_{i,j}(1, y_j) = 0 \ \forall y_j$ and if $y_i = 2, f_{i,j}(2, y_j) = C - A \ \forall y_j$). Thus we can represent these terms with f_i and f_j respectively.

2. If $f_i(1) < f_i(2)$, add an edge (s, y_i) with weight $f_i(2) - f_i(1) = C - A \geq 0$. Else, add edge (y_i, t) with weight $f_i(1) - f_i(2) = A - C \geq 0$. Repeat for j (thus having either weights $C - D$ or $D - C$).

3. Add an edge (y_i, y_j) with weight $B + C - A - D$ which is non-negative by assumption (submodularity).

Notice that according to this algorithm, we may obtain a graph such that there is no path from the source to the terminal (when $f_i(1) < f_i(2)\ \forall i$). In this case we can't use a graph-cut solver; however, in this case the solution is trivial.

4 Summary

In summary, the best method to infer the structured output is the following:

1. As long as K^D is small $\leq 10^5$, exhaustive search is fine. Exhaustive search becomes extremely difficult as D increases even slightly.

2. If there output structure is a tree (does not form cycles), the structured prediction can be determined using dynamic programming (DP) by solving one maximization at a time.

3. Integer Linear Programming (ILP) is pretty much a method to formulate an optimization program and should not be solved directly. In most cases, if it is possible to solve the ILP within a feasible time, it should also be feasible to use exhaustive search.

4. If DP cannot be used and the number of possible outputs is large, we may either solve the relaxed linear program by removing the integer constraints in ILP. Another method is to solve the dual, and obtain the primal solutions by substituting the solution to the dual (the messages or Lagrange multipliers).

5. If $K = 2$ but D is large (so that exhaustive search is infeasible), then we can resort to graph cut by making pairwise connections, constructing a graph according to the algorithm described above, and using a min-cut / max-flow solver (again, this is *very* different from max-cut).

Techniques for structured inference are summarized in table 1.

Inference Method	Advantage	Limitations
Exhaustive Search	Simple to implement and sufficient for many inference problems	Slow for large structures $D >> 1$
Dynamic Programming	Significantly reduces complexity	Only works for tree-structures
Integer Linear Programming	Works for any structures	Similar to Exhaustive Search due to integer constraints
Message Passing	Easy to solve (via convex programming)	Still may not obtain integer solutions (hence a sub-optimal solution)
Graph Cut	Has low complexity and is applicable to grid-structured outputs	Applicable to specific structures (we considered binary valued y's and pairwise relations)

Table 1: Comparison of Inference Methods

References

[1] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? Technical report, Ithaca, NY, USA, 2001.

[2] S. Nowozin and C. H. Lampert. Structured prediction and learning in computer vision. *Foundations and Trends in Computer Graphics and Vision*, 6(3-4), 2011.