Learning Structured Outputs via Kernel Dependency Estimation and Stochastic Grammars

Fabrizio Costa, Andrea Passerini, Paolo Frasconi

Machine Learning and Neural Networks Group
Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze, Italy
The Idea in a Nutshell

- Structured output prediction
- Output structures generated by stochastic grammar
- Output structure mapped into frequency of single production rules
- Predict mapped output by vectorial regression (KDE)
- Compute mapped output pre-image by Viterbi procedure
Kernel Dependency Estimation

Pre-requisites

- Input mapping \( \phi : \mathcal{X} \mapsto \mathcal{F}_\mathcal{X} \)
- Input Kernel \( \kappa(x, x') = \langle \phi(x), \phi(x') \rangle \)
- Output mapping \( \psi : \mathcal{Y} \mapsto \mathcal{F}_\mathcal{Y} \)
- Output Kernel \( \lambda(y, y') = \langle \psi(y), \psi(y') \rangle \)

Two-Stage Process

- Estimate output features \( g : \mathcal{X} \mapsto \mathcal{F}_\mathcal{Y} \)
- Compute pre-image \( \psi^{-1} : \mathcal{F}_\mathcal{Y} \mapsto \mathcal{Y} \)
Estimate $g : \mathcal{X} \rightarrow \mathcal{F}_Y$ given examples $\{(x_i, \psi(y_i))\}$.

Assume finite dimensionality $n_o$ for $\mathcal{F}_Y$.

Apply kernel ridge regression solving:

$$C = \Psi(y)(K + \gamma ml_m)^{-1}$$ (1)

with $K$ input kernel matrix, $\Psi(y)$ $n_o \times m$ matrix with columns $\psi(y)$, and solution given by:

$$g(x) = \sum_{i=1}^{m} c_i \kappa(x, x_i)$$ (2)

Efficient alternatives exist (e.g. maximum margin regression robot).
Pre-image calculation

- Estimate output mapping inversion \( \psi^{-1} : \mathcal{F}_Y \rightarrow Y \)

- Search space of structures for one with image nearest to \( g(x) \):

\[
f(x) = \arg \min_{y \in Y} \| g(x) - \psi(y) \|^2
\]  

(3)

- Using kernel ridge regression for \( g(x) \):

\[
\| g(x) - \psi(y) \|^2 = \lambda(y, y) - 2 \sum_{i,j=1}^{m} h_{ij} \lambda(y_i, y) \kappa(x_j, x)
\]  

(4)

\( h_{ij} \) being \( H = \{ h_{ij} \} = (K + \mu I)^{-1} \).

- Corinna et al. solved it by a graph theoretical algorithm in the case of output strings and \( k \)-gram output kernels.
Consider a stochastic grammar \( G(x) = \{ N, T, S, \Pi(x) \} \)

\( \Pi(x) \) are example dependent production rule probabilities (unknown on unseen examples)

The output feature mapping \( \psi(y) \) is a real vector encoding \( \Pi(x) \)

A probabilistic parser is used in the pre-image step to output the most probable parse given the estimated \( \Pi(x) \).
Production rules $r_{k\ell}$ have the form $A_k \mapsto \alpha_\ell$ with $A_k \in N$ and $\alpha_\ell \in (N \cup T \cup \{\epsilon\})^*$. 

Production rule $r_{k\ell}$ has an attached probability $\pi_{k,\ell}$ with constraints $\sum_\ell \pi_{k,\ell} = 1$ for each $k = 1, \ldots, |N|$. 

These probabilities are linked to the feature vector $\psi(y)$ by the softmax function:

$$
\pi_{k,\ell} = \frac{e^{\psi_{k,\ell}}}{\sum_{j=1} e^{\psi_{k,j}}}. 
$$

The feature estimation problem consists of solving a generalized linear model. 

A SCFG parser computes the most probable parse given the estimated $\Pi(x)$ by the inside-outside algorithm.
Prove that the algorithm can be applied where a standard SCFG parser fails.

Simulate PP-attachment ambiguity resolution:
- *eat the salad with the fork* ⇒ (VP (V eat) (NP a salad) (PP with a fork))
- *eat the salad with tomatoes* ⇒ (VP (V eat) (NP a salad (PP with tomatoes)))

Lexicalization (example dependent) is needed to resolve ambiguity.

Introduces a form of context-sensitiveness.
Ambiguity resolution simulated by the following grammar:

- $S \rightarrow ScS | NV$
- $V \rightarrow wNP | vNP$
- $N \rightarrow n | ncV$
- $NP \rightarrow nP | ncVP$
- $P \rightarrow pn$

- $w \rightarrow 5$
- $v \rightarrow 4$
- $n \rightarrow 2 | 3$
- $p \rightarrow 1$
- $c \rightarrow 0$

- Probabilities are uniform except for $S \rightarrow (.2) ScS | (.8) NV$
- Context-sensitiveness introduced by collapsing 'v' and 'w' in 'x'.
- A standard SCFG parser with probabilities estimated over the entire dataset cannot disambiguate.
Douglas Rohde’s Simple Language Generator (SLG) to randomly generate dataset.

Post-processed in two ways:

- *Natural* filtered out duplicate input sequences
- *Unique* filtered out sentences with identical representation in $F_Y$
Compared KDE-SCFG with standard SCFG

Used spectrum kernel with k-mers of size 2 to 5 to compute $\kappa$

use Collin’s `evalb` program to compute the bracketing F-measure and exact parse matching score.

Randomly split dataset in two sets (1,000 instances each)

Model selection on the first set (5-folds cv)

Performance evaluation on the second set (5-folds cv)
Results on the entire datasets and focused on short sequences (< 35 terminals) only.

KDE-SCFG significantly outperforms the SCFG parser ($p < .05$ in all pairwise comparisons).

<table>
<thead>
<tr>
<th>Filtering</th>
<th>Natural</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measure</td>
<td>F-score</td>
<td>Exact</td>
</tr>
<tr>
<td>SCFG&lt;35</td>
<td>86.4</td>
<td>10.3</td>
</tr>
<tr>
<td>SCFG</td>
<td>85.8</td>
<td>8.1</td>
</tr>
<tr>
<td>KDE-SCFG&lt;35</td>
<td>93.3</td>
<td>33.2</td>
</tr>
<tr>
<td>KDE-SCFG</td>
<td>91.5</td>
<td>26.1</td>
</tr>
</tbody>
</table>
Conclusions

- Novel solution to pre-image problem
- Use frequency of stochastic grammar production rules as output feature mapping
- Significantly outperformed standard SCFG parser on simplified NLP problem
- Further extensions are possible: e.g. use probabilistic ILP programs in place of SCFG.