Pyconstruct: Constraint Programming Meets Structured Prediction[*]

Paolo Dragone[1,2], **Stefano Teso**[3] and **Andrea Passerini**[1]
[1] University of Trento
[2] TIM-SKIL, Trento, Italy
[3] KU Leuven, Leuven, Belgium
paolo.dragone@unitn.it, stefano.teso@cs.kuleuven.be, andrea.passerini@unitn.it

Abstract

Constructive learning is the task of learning to *synthesize* structured objects from data. Examples range from classical sequence labeling to layout synthesis and drug design. Learning in these scenarios involves repeatedly synthesizing candidates subject to feasibility constraints and adapting the model based on the observed loss. Many synthesis problems of interest are non-standard: they involve discrete and continuous variables as well as arbitrary constraints among them. In these cases, widespread formalisms (like linear programming) can not be applied, and the developer is left with writing her own ad-hoc solver. This can be very time consuming and error prone. We introduce Pyconstruct, a Python library tailored for solving real-world constructive problems with minimal effort. The library leverages max-margin approaches to decouple learning from synthesis and constraint programming as a generic framework for synthesis. Pyconstruct enables easy prototyping of working solutions, allowing developers to write complex synthesis problems in a declarative fashion in few lines of code. The library is available at: http://bit.ly/2st8nt3

1 Introduction

Many real-world problems involve learning to synthesize (potentially novel) structures or solutions from examples. Applications range from classical problems like syntactic parsing and image segmentation to design tasks like layout synthesis [Yu *et al.*, 2011; Dragone *et al.*, 2016], interface optimization [Gajos and Weld, 2005], and drug design [Lavecchia, 2015]. Learning in this setting involves iteratively synthesizing structures according to the current model, usually via mathematical optimization, and updating the latter based on some estimate of the corresponding loss. Well-known models include structured-output SVMs [Tsochantaridis *et al.*, 2004] and conditional random fields [Sutton and McCallum, 2012].

A major issue with existing implementations is that they assume synthesis to be easily encodable in standard formalisms, like linear programming or dynamic programming. However, many problems of interest can not be reformulated in such a way. Indeed, they may involve discrete and continuous variables (e.g. type and position of furniture pieces in layout synthesis) and arbitrary constraints between them (non-overlap, design guidelines). In this case, the developer is left with the task of writing her own ad-hoc solver.

We introduce Pyconstruct, a Python library specifically tailored for non-standard constructive tasks. Our goal is to cut the effort needed for writing a working learner. Pyconstruct combines two ingredients. First, synthesis is expressed in MiniZinc [Nethercote *et al.*, 2007], a compact and general declarative language for constraint programming. MiniZinc enables writing non-standard synthesis problems in few lines of code. Other inference problems (such as separation [Joachims *et al.*, 2009]) are expressed in the same way, reusing code snippets whenever possible. MiniZinc comes with several state-of-the-art backends for combinatorial, numerical and mixed problems, e.g., OptiMathSAT [Sebastiani and Trentin, 2015], Gecode [Schulte *et al.*, 2010] and Gurobi [Gurobi Optimization, 2016]. Second, learning employs max-margin techniques, which decouple synthesis from learning. Therefore, the developer can change the synthesis problem without worrying about the learning algorithm at all. This speeds up prototyping solutions to non-standard constructive problems and adapting existing solutions to different settings. The library stems from several papers on constructive learning [Teso *et al.*, 2016; 2017a; 2017b; Dragone *et al.*, 2016; 2017; 2018a; 2018b].

2 Structured Prediction with Pyconstruct

We follow the usual structured output setting [Bakir *et al.*, 2007], where the goal is to induce a mapping $f : \mathcal{X} \to \mathcal{Y}$ from inputs $x \in \mathcal{X}$ (e.g., sentences, images, empty room layouts) to output structures $y \in \mathcal{Y}$ (tags, segments, furnished rooms). Synthesis, or inference, amounts to solving the optimization problem $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y) \rangle$. Here $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ is a joint input-output feature map and $\boldsymbol{w} \in \mathbb{R}^d$ is a vector of parameters to be estimated

[*]PD is a fellow of TIM-SKIL Trento and is supported by a TIM scholarship. This work has received funding under the European Union's Horizon 2020 research and innovation programme (ERC grant agreement No [694980] SYNTH: Synthesising Inductive Data Models and grant agreement No [732194] the QROWD project).

```
1  %%~~~~~~~~~~~~~~~~~~ domain.pmzn ~~~~~~~~~~~~~~~~~%%
2  %% Input: Length of the sequence and images
3  int: N; array[1..N, 1..9, 1..9] of {0, 1}: img;
4
5  %% Output: Sequence of symbols, digits "0" to "9"
6  %% plus and equal signs encoded with "10" and "11"
7  array[1 .. N] of var 0 .. 11: seq;
8
9  %% Indices of the two operators
10 array[1 .. 2] of var 2 .. N - 1: opr;
11 constraint increasing(opr);
12 constraint seq[opr[1]] == 10 /\ seq[opr[2]] == 11;
13 constraint count(seq, 10, 1) /\ count(seq, 11, 1);
14
15 array[1 .. 3] of var int: num = [
16     %% Numbers computed by summing powers of ten
17 ];
18
19 %% Numbers are positive and equation must be valid
20 constraint forall(i in 1 .. 3)(num[i] >= 0);
21 constraint num[1] + num[2] == num[3];
22
23 %% OCR features, correlate images to symbols
24
25 {% from 'pyconstruct.pmzn' import solve %}
26 {{ solve(problem, model) }}
27
28 %%~~~~~~~~~~~~~~~~~~ learn.py ~~~~~~~~~~~~~~~~~~~~%%
29 from pyconstruct import Domain, SSG
30 ssg = SSG(Domain('domain.pmzn')).fit(X, Y)
```

Figure 1: Example of Pyconstruct domain encoded with MiniZinc.

from data. Learning can be performed in a number of ways. We focus on max-margin approaches [Tsochantaridis *et al.*, 2005], since they only require an oracle able to solve inference (or related problems [Joachims *et al.*, 2009]) for the target structures. Existing implementations are usually limited to structures for which an efficient oracle is known, such as sequences, trees or graphs. In stark contrast, Pyconstruct makes the inference oracle *programmable*. In Pyconstruct the oracle is a solver-agnostic MiniZinc program, allowing the domain of objects to be defined independently from the inference algorithm, which can be chosen and plugged in at runtime. This layer of abstraction enables performing inference on non-standard objects and enforcing arbitrary constraints on them. Our library has two main components: (i) a Python learning framework implementing several state-of-the-art algorithms (e.g. SSG [Shalev-Shwartz *et al.*, 2011] and Block-Coordinate Frank-Wolfe [Lacoste-Julien *et al.*, 2013]); (ii) a constraint programming inference engine powered by MiniZinc [Nethercote *et al.*, 2007]. A MiniZinc file encodes the *domain* of the structured objects, which defines the input and output variables, the feasibility constraints, the feature vector, and methods for solving the different inference problems. Upon inference, Pyconstruct runs a MiniZinc-compatible solver on the domain file, specifying which inference problem to solve and which *model* to use. Models are objects holding the learned parameters, e.g. a linear model contains a vector $\boldsymbol{w}$ of weights. A model is usually the output of a *learner*, which estimates the model parameters from data using some learning algorithm. Learners in Pyconstruct are compatible with Scikit-learn [Pedregosa *et al.*, 2011], and can be used in conjunction with most of its utilities.

3 Example: OCR Equation Recognition

In this demo, we demonstrate how our library can be used to solve quite complex structured-output prediction problems in just a few lines of MiniZinc code. In particular, here we showcase a simple problem of handwritten equation recognition. In this task we have to recognize equations of the form $a + b = c$, where $a, b \geq 0$. The input is a sequence of N images, one per symbol. Notice that N is not fixed and depending on the number of digits of each number it may grow arbitrarily. We assume there are also two images for the "+" and "=" symbols, and that the equations are all valid. Encoding this prior knowledge inside a standard, Viterbi-like, inference algorithm over sequences would be quite verbose and difficult to optimize. Using Pyconstruct we can code this problem very easily[1], as shown in the `domain.pmzn` file in Figure 1. The length of the sequence N and the sequence of images (assumed to be 9 by 9 black or white pixels) are given as inputs. The output sequence `seq` is the only output variable and is encoded as a vector of length N of integers in $[0, 11]$, where 0 to 9 represent the digits themselves, while 10 and 11 represent $+$ and $=$ respectively. In line 10 we define an array containing the indices of the two operators. The following three lines encode what we know about those indices, i.e. their order and their correspondence with their respective symbols in the sequence. Also, the sequence must contain exactly one $+$ and one $=$ (line 13). Next, we define the three actual numbers as $n_i = \sum_{0 \leq j < m_i} 10^j \cdot d_{i,j}$, where m_i is the number of digits in number i and $d_{i,j}$ is the j-th digit of the i-th number, from the least to the most significant digit (omitted in Figure 1 for brevity). Finally, we constrain the numbers to be positive and the equation to be valid (lines 20 and 21). For space limitations, we also omitted the code describing the feature array. This should be an array of features that correlate the images to the symbols, e.g. [Taskar *et al.*, 2004]. The last two lines contain templating code that Pyconstruct compiles to a proper MiniZinc solve statement, depending on the inference problem to be solved and the current model. After defining the MiniZinc file encoding the objects domain, one can estimate a structured-output model over it. All is needed is to instantiate a `Domain` providing the MiniZinc file in Figure 1, instantiate a `Learner` and `fit` it with the data. This takes exactly two lines of Python code, as shown in the `learn.py` file in Figure 1, where we use an `SSG` learner and fit it with some data `(X, Y)`. As an example, we used Pyconstruct to solve the above formula recognition task on a small dataset extracted from the ICFHR'14 CROHME competition data[2]. We used an `SSG` learner and compared the performance of the domain in Figure 1 against a similar domain lacking the prior knowledge encoded with the MiniZinc constraints. After training the algorithm with 800 samples, the constrained model achieves an average Hamming loss on the predicted sequences over the test set (200 samples) of 0.102, while the unconstrained one stops at 0.373 ($p < 10^{-30}$). The learning curves are also widely separated: the training losses are on average 22% lower for the constrained domain ($p < 10^{-6}$), confirming that the constraints help learning with less data.

[1]Code and data available at http://bit.ly/2LX0sMO

[2]Data available at http://bit.ly/2J7Dh4v

References

[Bakir *et al.*, 2007] Gükhan H. Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan. *Predicting Structured Data*. MIT Press, 2007.

[Dragone *et al.*, 2016] Paolo Dragone, Luca Erculiani, Maria Teresa Chietera, Stefano Teso, and Andrea Passerini. Constructive layout synthesis via coactive learning. In *Constructive Machine Learning workshop, NIPS*, 2016.

[Dragone *et al.*, 2017] Paolo Dragone, Stefano Teso, and Andrea Passerini. Constructive preference elicitation. *Frontiers in Robotics and AI*, 4:71, 2017.

[Dragone *et al.*, 2018a] Paolo Dragone, Stefano Teso, Mohit Kumar, and Andrea Passerini. Decomposition strategies for constructive preference elicitation. In *AAAI*, 2018.

[Dragone *et al.*, 2018b] Paolo Dragone, Stefano Teso, and Andrea Passerini. Constructive preference elicitation over hybrid combinatorial spaces. In *AAAI*, 2018.

[Gajos and Weld, 2005] Krzysztof Gajos and Daniel S Weld. Preference elicitation for interface optimization. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 173–182. ACM, 2005.

[Gurobi Optimization, 2016] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2016.

[Joachims *et al.*, 2009] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.

[Lacoste-Julien *et al.*, 2013] Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. Block-coordinate frank-wolfe optimization for structural svms. In *ICML 2013 International Conference on Machine Learning*, pages 53–61, 2013.

[Lavecchia, 2015] Antonio Lavecchia. Machine-learning approaches in drug discovery: methods and applications. *Drug discovery today*, 20(3):318–331, 2015.

[Nethercote *et al.*, 2007] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Schulte *et al.*, 2010] Christian Schulte, Guido Tack, and Mikael Z Lagerkvist. Modeling and programming with gecode. *Schulte, Christian and Tack, Guido and Lagerkvist, Mikael*, (2015), 2010.

[Sebastiani and Trentin, 2015] Roberto Sebastiani and Patrick Trentin. Optimathsat: a tool for optimization modulo theories. In *International Conference on Computer Aided Verification*, pages 447–454. Springer, 2015.

[Shalev-Shwartz *et al.*, 2011] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[Sutton and McCallum, 2012] Charles Sutton and Andrew McCallum. An introduction to conditional random fields. *Found. Trends Mach. Learn.*, 4(4):267–373, April 2012.

[Taskar *et al.*, 2004] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *Advances in neural information processing systems*, pages 25–32, 2004.

[Teso *et al.*, 2016] Stefano Teso, Paolo Dragone, and Andrea Passerini. Structured feedback for preference elicitation in complex domains. In *BeyondLabeler Workshop at IJCAI 2016*, 2016.

[Teso *et al.*, 2017a] Stefano Teso, Paolo Dragone, and Andrea Passerini. Coactive critiquing: Elicitation of preferences and features. In *AAAI*, 2017.

[Teso *et al.*, 2017b] Stefano Teso, Roberto Sebastiani, and Andrea Passerini. Structured learning modulo theories. *Artificial Intelligence*, 244:166–187, 2017.

[Tsochantaridis *et al.*, 2004] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*, page 104. ACM, 2004.

[Tsochantaridis *et al.*, 2005] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 6:1453–1484, 2005.

[Yu *et al.*, 2011] Lap Fai Yu, Sai Kit Yeung, Chi Keung Tang, Demetri Terzopoulos, Tony F Chan, and Stanley J Osher. Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (TOG)-Proceedings of ACM SIGGRAPH 2011, v. 30, no. 4, July 2011, article no. 86*, 2011.