srlearn: A Python Library for Gradient-Boosted Statistical Relational Models

Alexander L. Hayes
ProHealth Lab
Indiana University Bloomington
hayesall@iu.edu

Abstract

We present `srlearn`, a Python library for boosted statistical relational models. We adapt the scikit-learn interface to this setting and provide examples for how this can be used to express learning and inference problems.

Introduction

Traditional machine learning systems have generally been built as command line applications or as graphical user interfaces (Hall et al. 2009). Both have advantages, but offer fewer solutions when data acquisition, preprocessing, and model development must occur together. Systems such as scikit-learn, TensorFlow, Pyro, and PyTorch solve this problem by embedding data cleaning and model development as steps within general-purpose languages (Pedregosa et al. 2011; Abadi et al. 2016; Bingham et al. 2019; Paszke et al. 2017). This has also made open source implementations available to both experts and non-experts, providing each the tools to develop models.

Statistical Relational Learning (SRL) models have unique concerns, often inherited from underlying logical systems. This requires a data representation beyond fixed-length feature vectors, and a language bias to constrain the hypothesis space. By embedding both operations in a manner that machine learning researchers and practitioners may already be familiar with, we hope to speed up development time for SRL practitioners, and provide a more user-friendly experience for data scientists and the wider machine learning community—many of whom are not experts in SRL.

API Design in Machine Learning

The scikit-learn package (Pedregosa et al. 2011) has been influential for its consistent application programming interface (API) across a variety of machine learning models. In scikit-learn, an algorithm type (e.g. linear support vector classification) is implemented as a class. An *estimator* is an instance of an algorithm type whose hyperparameters have been set upon object construction. A *predictor* is an estimator that has been `fit` (i.e. trained) to a dataset, and is ready to `predict` (e.g. classify) new data instances. Although the estimation and prediction functions are logically distinguished in two separate protocols, it is generally a single class that implements both a learning algorithm and the model for applying the parameters to new data.

The aforementioned standard approach thus comprises configuring model hyperparameters, fitting training data, and predicting test data. However, scikit-learn has since developed into a full-fledged ecosystem that also services related functions in the modeling workflow: model selection, hyperparameter tuning, and model validation. Furthermore, multiple offshoots of scikit-learn have emerged to tackle more specialized challenges, including imbalanced datasets (Lemaître, Nogueira, and Aridas 2017), generalized linear models (Blondel and Pedregosa 2016), and metric learning (de Vazelhes et al. 2019).

These offshoots still fit within the framework of only requiring inputs, outputs, and hyperparameters. But while this has been influential while designing APIs for classic statistical learning methods, it could also be a limitation when extending the API to incorporate the specific needs of models from other learning paradigms (Buitinck et al. 2013). Learning within frameworks designed for graphical models, active learning, or reinforcement learning typically requires the user to specify something outside of inputs and outputs. Graphical models require statistical independence assumptions to either be set by hand or inferred via structure learning. Active learning requires human intervention. Reinforcement learning needs a simulator.

Extending the API to handle new paradigms should ideally meet two goals: (1) *expressiveness* to describe what the user wants to achieve, and (2) *complimentarity* to what users are already familiar with.

`srlearn`

We propose a simple extension to the scikit-learn API for representing statistical relational models while staying close to our two goals. Specifically we incorporate a `Background` object and a `Database` object.

The `Background` object incorporates knowledge about relationships to constrain model search space, currently expressed in the language of "modes" (Srinivasan 2000). This is then provided to the statistical relational estimator.

The `Database` object generalizes inputs as being composed of positive examples, negative examples, and facts about the world—each expressed as Prolog predicates.

```python
from srlearn.rdn import BoostedRDN
from srlearn import Background
from srlearn import example_data

bk = Background(
  modes=[
    "friends(+person,-person).",
    "friends(-person,+person).",
    "cancer(+person).",
    "smokes(+person).",
  ]
  use_std_logic_variables=True,
)

clf = BoostedRDN(
  background=bk,
  target="cancer",
)

clf.fit(example_data.train)
clf.predict_proba(example_data.test)
```

Figure 1: Learning and inference on toy databases for a smokes-friends-cancer domain. `example_data.train` and `example_data.test` are `Database` objects.

A statistical relational estimator may then be described in the same language as a standard scikit-learn estimator that also incorporates background knowledge to constrain the hypothesis space, and learn on a database of predicates rather than vectors. Currently we have focused on incorporating methods from BOOSTSRL, a Java tool for learning relational dependency networks and Markov logic networks via gradient boosting (Natarajan et al. 2018). Figure 1 shows how modules from `srlearn` can be put together to learn on a built-in data set, then make predictions on a test database.

Development

`srlearn` is developed as an open source project on GitHub[1] and is distributed under the terms of the GNU General Public License v3.0 (GPL-3.0). Within the code, we have taken several measures to aid its maintenance. This includes formatting conventions (`black`, `pycodestyle`), linting (`pylint`), and running the main branch and all pull requests through static analysis (`lgtm`).

We also maintain a test suite to compare each build against previous versions. Tests run on Linux and Windows machines each time the code is pushed to GitHub; metrics track (1) that all tests pass, and (2) that a sufficient code coverage is maintained. At the time of writing, all tests pass (results meet expectations), and code coverage is at 100% (every line of code is visited during testing). Perfect coverage often grows unrealistic as projects grow, but we aim to keep it above 90% while passing all tests.

Finally, we maintain documentation[2] to help acclimate users to the code base; this includes user guides with narrative documentation and examples motivating specific tasks.

[1] https://github.com/hayesall/srlearn/

[2] https://srlearn.readthedocs.io

	`srlearn` (Python/Java)	BOOSTSRL (Java/Shell)
WebKB	4.2 (0.5)	4.9 (0.3)
IMDB	10.2 (1.1)	13.0 (1.3)
UWCSE	17.5 (1.4)	18.3 (1.7)

Table 1: Seconds elapsed while learning a Boosted RDN on three benchmark data sets. Mean (and standard deviation) are calculated over ten runs. Small differences in times may also be influenced by small differences in measurement: epoch time (Bash) and `perf_time` (Python).

Experiments

We expect a small overhead due to the Python interpreter and data structures at runtime; but since the core algorithms borrow heavily BOOSTSRL's Java implementations, we expect this overhead to be negligible compared to the time spent during learning. To evaluate this, we compare runtime in seconds on standard benchmark data using the BOOSTSRL command line interface and the `srlearn` API. We hold the modes and hyperparameters fixed, then record the time taken while learning a boosted RDN with the `srlearn` and BOOSTSRL systems on three benchmark data sets. Table 1 shows the time averaged over ten runs of each, which we use to conclude that the time differences are indeed negligible.[3]

Conclusion

It is possible that the imperative programming style here is not ideal for SRL models—the underlying logic formalism is often better expressed through declarative approaches, which have further been suggested as ways to unify software development with learning systems (Kordjamshidi, Roth, and Kersting 2018).

Nonetheless, many learning frameworks have been built around the Python ecosystem. Programming abstractions such as the one presented here may therefore be an important step toward bridging the gap between SRL and neural approaches by providing developers the tools to more easily work with both in a common environment.

In the future, we intend on extending the modeling language with more methods that have been successful within SRL—such as learning with advice (Odom and Natarajan 2018), incorporating a relational database for learning and inference (Malec et al. 2017), and incorporating SRL methods such as Probabilistic Soft Logic (Bach et al. 2015) or Conditional Random Fields (Sutton and McCallum 2007).

Acknowledgements

ALH is supported through Indiana University's "Precision Health Initiative" (PHI) Grand Challenge. ALH would like to thank Sriraam Natarajan, Travis LaGrone, and members of the StARLinG Lab at the University of Texas at Dallas.

[3] Scripts for reproducing this table is available on GitHub: https://github.com/hayesall/srlearn-StarAI-2020-workshop

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; Kudlur, M.; Levenberg, J.; Monga, R.; Moore, S.; Murray, D. G.; Steiner, B.; Tucker, P.; Vasudevan, V.; Warden, P.; Wicke, M.; Yu, Y.; and Zheng, X. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 265–283. Savannah, GA: USENIX Association.

Bach, S. H.; Broecheler, M.; Huang, B.; and Getoor, L. 2015. Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research (JMLR)*.

Bingham, E.; Chen, J. P.; Jankowiak, M.; Obermeyer, F.; Pradhan, N.; Karaletsos, T.; Singh, R.; Szerlip, P.; Horsfall, P.; and Goodman, N. D. 2019. Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research* 20(28):1–6.

Blondel, M., and Pedregosa, F. 2016. Lightning: large-scale linear classification, regression and ranking in Python.

Buitinck, L.; Louppe, G.; Blondel, M.; Pedregosa, F.; Mueller, A.; Grisel, O.; Niculae, V.; Prettenhofer, P.; Gramfort, A.; Grobler, J.; Layton, R.; VanderPlas, J.; Joly, A.; Holt, B.; and Varoquaux, G. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 108–122.

de Vazelhes, W.; Carey, C.; Tang, Y.; Vauquier, N.; and Bellet, A. 2019. metric-learn: Metric Learning Algorithms in Python. Technical report, arXiv:1908.04710.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The weka data mining software: An update. *SIGKDD Explor. Newsl.* 11(1):10–18.

Kordjamshidi, P.; Roth, D.; and Kersting, K. 2018. Systems ai: A declarative learning based programming perspective. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 5464–5471. International Joint Conferences on Artificial Intelligence Organization.

Lemaître, G.; Nogueira, F.; and Aridas, C. K. 2017. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research* 18(17):1–5.

Malec, M.; Khot, T.; Nagy, J.; Blask, E.; and Natarajan, S. 2017. Inductive logic programming meets relational databases: Efficient learning of markov logic networks. In Cussens, J., and Russo, A., eds., *Inductive Logic Programming*, 14–26. Springer International Publishing.

Natarajan, S.; Odom, P.; Khot, T.; Kersting, K.; and Shavlik, J. 2018. Human-in-the-loop learning for probabilistic programming. *Proceedings of the Inaugural International Conference on Probabilistic Programming*.

Odom, P., and Natarajan, S. 2018. Human-guided learning for probabilistic logic models. *Frontiers in Robotics and AI* 5:56.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

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

Srinivasan, A. 2000. The Aleph Manual. Technical report, Computing Laboratory, Oxford University, Oxford, UK. https://www.cs.ox.ac.uk/activities/programinduction/Aleph/.

Sutton, C., and McCallum, A. 2007. *An Introduction to Conditional Random Fields for Relational Learning*. Cambridge, Massachusetts: MIT Press. chapter 4, 93–127. in: Introduction to Statistical Relational Learning.