Flexible State-Merging for Learning (P)DFAs in Python

Christian Hammerschmidt FIRSTNAME.LASTNAME@UNI.LU
Benjamin Loos LASTNAME.FIRSTNAME@GMAIL.COM
Radu State FIRSTNAME.LASTNAME@UNI.LU
Thomas Engel FIRSTNAME.LASTNAME@UNI.LU
Interdisciplinary Centre for Security, Reliability and Trust
4ue Alphonse Weicker, L-2721, Luxembourg

Sicco Verwer S.E.VERWER@TUDELFT.NL
TU Delft, Mekelweg 4, 2628 CD, Delft, The Netherlands

Abstract

We present a Python package for learning (non-)probabilistic deterministic finite state automata and provide heuristics in the red-blue framework. As our package is built along the API of the popular `scikit-learn` package, it is easy to use and new learning methods are easy to add. It provides PDFA learning as an additional tool for sequence prediction or classification to data scientists, without the need to understand the algorithm itself but rather the limitations of PDFA as a model. With applications of automata learning in diverse fields such as network traffic analysis, software engineering and biology, a stratified package opens opportunities for practitioners.

Keywords: machine learning, grammar inference, automaton learning, PDFA inference

1. Introduction

Automata models are highly relevant in computer science: State-based models are frequently used to specify protocols and software behavior prior to implementation. In these cases, while not providing the same degree of generality as function approximating models such as neural networks, automaton inference allows to recover a model similar to the specification and offers advantages as an interpretable white box model. With the growth of data science as a discipline of applied machine learning, a large software ecosystem ranging from big data frameworks like Hadoop and Spark[1] to machine learning libraries like `scikit-learn` and `weka`[2] was created. It contributes to an implicit standardization and enables practitioners to leverage a wide range of algorithms without the need to re-implement or understand the details of algorithms. Unfortunately, very few frameworks provide tools for automaton inference. `scikit-learn` for instance removed their HMM inference algorithms from the core package[3]. Other libraries to learn automata are not well-integrated in today's data science landscape. For a review of other libraries, see Cottone et al.. Our goals are therefore twofold. We want to commoditize automaton learning and provide easy-to-use algorithms to a wide audience. Moreover, we want to enable reproducible research by providing a

1. `http://hadoop.apache.org/` and `http://spark.apache.org/`
2. Pedregosa et al. (2011), `http://scikit-learn.org/` and `http://www.cs.waikato.ac.nz/ml/weka/`
3. `http://scikit-learn.org/stable/faq.html#adding-graphical-models`

package to not only share code, but also results and work flows. We chose Python, which has seamless integration with Jupyter notebooks[4].

2. Flexible State-Merging in `dfasat`

In PDFA learning using a state-merging approach, see e.g. Verwer et al. (2014), a set S_+ of observed behaviors encoded as words over an alphabet Σ called the *input sample* is given. The goal is to find a (non-unique) *smallest* probabilistic deterministic finite state automaton (PDFA) A that is *consistent* with S_+. A PDFA is considered consistent with S_+ if it satisfies a type of Markov property, i.e., for every prefix s from S_+ that reaches the same state q in A, the sample probabilities of future suffixes $P(s' \mid s) = \mathit{count}(ss')/\mathit{count}(s)$ of the states are not significantly different. The size of a PDFA is measured by its number of states. Although they are similar, there is an important difference between hidden Markov models (HMMs) and PDFAs, namely that PDFAs are deterministic: For any state, there can be at most one transition for every possible input symbol. This adds a degree of interpretability (a simple state representation), and perhaps improved learnability (in theory, but debatable in practice Verwer et al. (2014)), at the cost of representational power Dupont et al. (2005).

One of the most popular algorithms has for a long time been ALERGIA Carrasco and Oncina (1994), or one of its variants such as MDI Thollard et al.. They are implementations of the basic state merging method, which is also a popular method for learning non-probabilistic automata. The starting point for state merging algorithms is the construction of a tree-shaped PDFA A from the input sample S_+. This is called augmented prefix tree acceptor (APTA) and contains all samples from S_+ in a directed graph, using the symbols of the samples in S_+ as labels for the edges. Two samples from S_+ share a path if they share a prefix. The state merging algorithm reduces the size of the automaton iteratively by reducing the tree through merging pairs of states in A, and forcing the result to be deterministic. Merges generalize the model beyond the samples from the training set. The key difference between the different state-merging approaches is the *heuristic* used to decide which pairs are best to merge, and the *consistency test* used to decide whether a merge is valid. ALERGIA Carrasco and Oncina (1994) performs merges unless a statistical test based on the Hoeffding bound suggests that the Markov property is violated. In the MDI algorithm this test is replaced by a function that computes the Kullback-Leibler (KL) divergence between the merged model and the original input sample. Other ALERGIA variants such as Young-Lai and Tompa (2000); Carrasco et al. (2001); Verwer et al. (2010) differ in the type of test performed, some learning other types of automata than PDFA.

`dfasat` Heule and Verwer (2013) is another state-merging style algorithm for learning non-probabilistic automata. In addition to greedy state merging, it uses SAT-solvers (see, e.g., Biere et al. (2009)) to solve the learning problem exactly. It uses a variant of the popular evidence-driven state-merging algorithm (EDSM) using the red-blue framework Lang (1998), but with a different (overlap driven) merge heuristic. By maintaining a core of already identified merged states, the red-blue framework reduces the number of candidate merges to check. Since this heuristic uses only information from the positive sample S_+, it can be used for learning probabilistic automata. We are currently extending the `dfasat` framework to be flexible in the used heuristic and consistency test such that it is possible to

4. `http://jupyter.org/`

implement new heuristics with very little effort: consistency tests only requires adding a single C++ file to the framework, inheriting essential functionality from base classes provided by us. We provide ALERGIA, MDI, the likelihood-ratio test from RTI+, EDSM, a learner for Mealy machines, and regression automata for time-series with the framework, making use of the same efficient union/find data structures and the exact solution via satisfiability.

3. Scikit-learn and `dfasat-python`

Model learners in `scikit-learn` are encapsulated in *estimator* objects. After instantiating a learner and initializing the object, its *fit* method can be called to train on data and labels passed to it, and *predict* can be called on new data to obtain predictions. This flow is consistent across a wide range of different learners, and allows the user to exchange the type of learner with minimal changes to their application. The following listing compares a SVM with our PDFA learner.

```
from sklearn import svm
# get training samples and labels
X_samples, Y_labels = get_data()
# initialize classifier
clf = svm.SVC(gamma=0.001,
        C=100)


# learn and predict
clf.fit(X_samples, Y_labels)
clf.predict(sequence)
```

```
from dfasat import DFASATEstimator
# get training samples and labels
X_samples, Y_labels = get_data()
# initialize classifier
estimator = DFASATEstimator(
        hName="alergia",
        hData="alergia_data",
        tries=1, state_count=25)
# learn and predict
estimator.fit(X_samples, Y_labels)
estimator.predict(sequence)
```

Similarly, a bagging classifier takes an estimator class and its parameters, and returns and object to call *fit* and *predict* on. Currently, we average all estimator predictions.

```
bag = BaggingClassifier(estimator=DFASATEstimator, number=50, random_seed=
    True, random_counts=[5, 15, 25], ...)
fit = bag.fit(train_data_x, train_data_y, subset=True)
```

A core goal of the package development was minimal interference with the existing codebase and ideally an encapsulation that is totally transparent to the developers modifying the underlying C++ code of `dfasat`. Using `boost.python`[5], we exposed the key classes to Python and wrote the estimator objects as wrappers.

4. Participation in SPiCE and Results

We used the competition to test our package. The goals were to identify what is needed for a good user experience during fast model iteration and learning ensembles. The assumed user can understand and interpret PDFA model performance, but has only little knowledge of state-merging algorithm and its heuristics. We assume that all problem sets were generated by a PDFA model. During the competition, we primarily focused on a modified ALERGIA implementation as outlined in Appendix B. We later on also tried likelihood and overlap heuristics provided by `dfasat`, but due to time constraints did not follow up on their performance. For each problem set we first ran one instance of the heuristic with default parameters. Table 2 in the appendix shows the results. For problems where we obtained

5. http://www.boost.org/doc/libs/1_61_0/libs/python/doc/html/index.html

competitive scores with fast run-time $(0, 5, 6, 9, 14)$ we ran a bagging classifier with 50 and 100 instances using `dfasat`'s random greedy merge strategy. For all other problems we submitted the baseline, but to smooth the models, we each trained an ensemble on random subsets. We did not customize the merge heuristic nor did we do a deeper analysis of the parameters of the algorithm used. We first used default parameters, and then increased and decreased the values for the *state_count*, *transition_count*, and *extra_parameter*.

5. Conclusion and Outlook

Our `dfasat-python` package can be used for fast model iteration within Jupyter notebooks for easy collaboration and sharing of code and work flows. We are working on integrating custom merge heuristics written in Python. This is possible using the same `boost.python` library we used to expose the C++ objects. Once this feature is integrated, our package can be used for fast prototyping and development of custom merge heuristics.

Acknowledgments

This work was partially funded by the FNR AFR-PPP grant *PAULINE*, and STW VENI project 13136 *MANTA* and NWO project 62001628 *LEMMA*.

References

A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh. Handbook of satisfiability: Volume 185 frontiers in artificial intelligence and applications, 2009.

R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. pages 139–152. Springer-Verlag, 1994.

R. C. Carrasco, J. Oncina, and J. Calera-Rubio. Stochastic inference of regular tree languages. *Machine Learning*, 44(1):185–197, 2001.

P Cottone, M Ortolani, and G Pergola. Gi-learning: An optimized framework for grammatical inference.

P. Dupont, F. Denis, and Y. Esposito. Links between probabilistic automata and hidden markov models: Probability distributions, learning models and induction algorithms. *Pattern Recogn.*, 38 (9):1349–1371, September 2005.

M. JH Heule and S. Verwer. Software model synthesis using satisfiability solvers. *Empirical Software Engineering*, 18(4):825–856, 2013.

et al. Lang, K. J. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In *Proceedings of the 4th International Colloquium on Grammatical Inference*, ICGI '98, pages 1–12, London, UK, UK, 1998. Springer-Verlag.

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.

F. Thollard, P. Dupont, C. de la Higuera, et al. Probabilistic dfa inference using kullback-leibler divergence and minimality.

S. Verwer, M. de Weerdt, and C. Witteveen. *A Likelihood-Ratio Test for Identifying Probabilistic Deterministic Real-Time Automata from Positive Data.* Springer Berlin Heidelberg, 2010.

S. Verwer, R. Eyraud, and C. De La Higuera. Pautomac: a probabilistic automata and hidden markov models learning competition. *Machine learning*, 96(1-2):129–154, 2014.

M. Young-Lai and F. Tompa. Stochastic grammatical inference of text database structure. *Machine Learning*, 40(2):111–137, 2000.

Appendix A. Installing the Package

We are currently preparing the package for publication by adding extensive documentation and suitable class- and integration tests. The package is already available for installation via `pip` from the testing server. The source code of the package is be available in the python branch of the `dfasat` repository. The following listing has the install instructions to build `dfasat` and the Python interface from scratch. It has been tested thoroughly in Ubuntu systems.

```
# Install dependencies
sudo apt-get install mercurial python-virtualenv
sudo apt-get install libatlas-base-dev gfortran libblas-dev libgsl0-dev
    castxml libpopt-dev liblapack-dev libboost-python-dev python3-dev

# (Optional) Create and activate a python3 virtualenv to contain the project
virtualenv test -p python3
cd test
. bin/activate

# Install py++
hg clone https://bitbucket.org/ompl/pyplusplus
cd pyplusplus
python3 setup.py install
cd ..
rm -rf pyplusplus

# Install pygccxml
git clone https://github.com/gccxml/pygccxml.git
cd pygccxml
python3 setup.py install
cd ..
rm -rf pygccxml

# Install more dependencies
pip3 install numpy scipy

# Compile and install dfasat
# This step needs more than 2GB RAM
pip3 install --extra-index-url https://testpypi.python.org/pypi dfasat
```

Appendix B. A Modified ALERGIA

For the SPiCE competition, we made several modifications to the basic ALERGIA algorithm. Firstly, by default, ALERGIA attempts merges in a fixed (e.g., lexicographical or shallow-first) order. This makes is easier to prove theoretical guarantees such as identification in the limit Carrasco and Oncina (1994), but in our opinion makes little sense in practice. It is much more natural to order the merges based on the amount of available information, or on a merge heuristic such as in EDSM Lang (1998). In our current version of ALERGIA, we try all possible merges (in a red-blue fashion) and simply count the number of merges performed as a heuristic, which is used in the `dfasat` randomized greedy search strategy, see Heule and Verwer (2013). A second modification we made is a new way of dealing with low frequency symbols. In ALERGIA, the Hoeffding bound test is performed

symbol	a	b	c	d	e	f	g	h
count state 1	2	2	2	2	0	0	0	0
count state 2	0	0	0	0	2	2	2	2

Table 1: Two states that should be identified and treated differently.

iteratively on all symbol counts. When these counts are very small, this test will succeed for all symbols, also for instance, when the data on symbols $a - h$ are distributed in the following way: In state 1. a, b, c, d have counts of 2 and e, f, g, h occur 0 times. In state 2, the groups are switched, i.e. a, b, c, d don't occur and e, f, g, h each occur twice. Table 1 summarizes the situation. The two states are clearly different, but the individual symbol distributions (0 vs. 2 counts) all provide insufficient data to draw this conclusion. A standard method to deal with low frequencies is to pool (bin) then together and treat them as one single symbol. In this case, this will give a pool with a count of 8 in both states, again no indication that the two states are different. Our idea of dealing with this problem is to bin these counts instead into 2 symbols: one where the counts in state 1 are smaller than those in state 2, and one where the counts in state 2 are smaller than those in state 1. We then apply the Hoeffding bound unmodified to the two pools.[6]

Appendix C. Score Overview

Set	ALERGIA	3-gram	spectral	likelihood	overlap	Submitted	Score	Rank	Best
1	0.841	0.843	**0.874**	0.841	0.841	spectral	0.879	4	0.918
2	0.823	0.818	**0.872**	0.799	0.768	spectral	0.874	5	0.920
3	0.778	0.776	**0.828**	0.790	0.722	spectral	0.825	7	0.886
4	0.370	**0.538**	0.470	0.400	0.439	3-gram	0.528	5	0.608
5	**0.551**	0.527	0.361	/	/	$ALERGIA$	0.555	8	0.810
6	0.650	**0.675**	0.631	0.531	0.597	$ALERGIA_{50}$	0.724	7	0.860
7	0.337	**0.442**	0.367	/	/	3-gram	0.440	7	0.785
8	0.512	**0.591**	0.535	0.494	0.507	3-gram	0.597	4	0.657
9	**0.929**	0.859	0.818	0.867	0.847	$ALERGIA_{50}$	0.948	2	0.963
10	0.284	**0.413**	0.302	/	/	3-gram	0.396	7	0.552
11	0.301	**0.389**	0.379	/	/	3-gram	0.372	9	0.544
12	0.654	**0.700**	0.575	/	/	3-gram	0.699	7	0.811
13	0.299	**0.436**	0.372	/	/	4-gram	0.455	4	0.588
14	0.306	0.339	**0.359**	0.399	0.326	$ALERGIA_{100}$	0.379	7	0.465
15	**0.265**	0.251	0.221	/	/	spectral	0.279	5	0.298

Table 2: Individual scores for different heuristics, and the final submission score. A slash / indicates that the run-time was too long for us. The index number indicates the number of instances in the ensemble using `dfasat`'s random greedy method on randomly chosen 2/3 of the training words.

6. Obviously, this test will fail more often than it should due to the biased binning strategy, we are still working on the mathematics needed to correct for this bias.