M5 / M5’ model trees in python with \texttt{m5py} compliant with scikit-learn

PyConDE PyData Berlin – April 12\textsuperscript{th}, 2022

Sylvain Marié - Senior Group Expert
Schneider Electric Artificial Intelligence Hub
Introduction

Power distribution devices
Plug’n play network protocols
Service-oriented components

Offices, schools, hotels…
Energy efficiency tracking
Microgrid campus

Critical processes
Condition monitoring
Predictive maintenance

Water distribution networks
Demand forecasting

Industry, Manufacturing
Advanced process control
Energy efficient processes

We are hiring data scientists & data engineers!
M5 / M5’ ? Once upon a time…

v1 (2012-2016, RIP)

Version control
Analytics language
Machine Learning
Web service wrapper
Model storage
Cloud host

Our first analytics-as-a-service cloud platform!

M5P model was here 😊

(VMs = IaaS)
M5 / M5’ ? Once upon a time…

Our first analytics-as-a-service cloud platform!

**v1 (2012-2016, RIP)**

- Version control
- Analytics language
- Machine Learning
- Web service wrapper
- Model storage
- Cloud host

(M5P model was here 😊)

...but not there 😞

**v2 (2016-2020)**

- (VMs = IaaS)

PyCon.DE & PyData 2022 - Sylvain Marié, Schneider Electric | Page 4

Principles (1):

- Grow a regression tree model (as in CART) by splitting the dataset (“divide-and-conquer”), using impurity = std(y). Stop the splitting process when impurity or nb samples is too small.

- Start by placing a constant predictor at each leave, as in CART.

New data (x1=21, x2=13, x3=1)

Principles (2):
- For all nodes in the tree, train a linear model on the samples reaching that node.
- Simplify it by greedily removing parameters until the performance-vs-complexity tradeoff (an “adjusted” MAE) stops decreasing.
- Prune from bottom-up: everytime a linear model at a node is better than the combination of its two children, cut the subtree

\[ LM: y = a_1 \times x_1 + b \]

\[ err_{LM} = MAE_{LM} \times \frac{n + \nu}{n - \nu} \]

Principles (2):
- For all nodes in the tree, train a linear model on the samples reaching that node.
- Simplify it by greedily removing parameters until the performance-vs-complexity tradeoff (an “adjusted” MAE) stops decreasing.
- Prune from bottom-up: everytime a linear model at a node is better than the combination of its two children, cut the subtree.

\[ LM: y = a_1 \times x_1 + b \]

\[ err_{LM} = MAE_{LM} \times \frac{n + v}{n - v} \]

Principles (3):
- Smooth recursively when predicting, in order for nodes with few samples to be corrected by their parent(s)

\[
PV(S) = \frac{n_i \times PV(S_i) + k \times M(S)}{n_i + k}
\]
M5 – why?

Piecewise linear: model complex nonlinear relationships while preserving interpretability.

Constant leaves

Linear model leaves
M5’ (prime)


- Brings clarification + a reference implementation to the initial idea
- 2 improvements: handling categorical attributes (enumerations) and missing values
- Results show how the two improvements behave on datasets with many categorical/missing
- An additional improvement seems to be provided (“pruning factor”) but without explanation in the paper, apart from “we introduce a modification which allows the tree size to be reduced dramatically”. From the code implemented in Weka, it could be this new PF (default=2):

\[
err_{LM} = MAE_{LM} \times \frac{n + PF \times v}{n - v}
\]
Timeline

• First implementation in 2016 (internal project)
• PR opened at scikit-learn in 2019 (scikit-learn#13732)
  • Stuck since, mostly because of overall complexity and documentation quality.
  • Still, several community members explicitly stated their interest.
  • It seems to me that this is interesting for educational purposes too?

Today’s proposal

- I put something here https://smarie.github.io/python-m5p/
- And made a first release (0.3.0) 😊
Implementing `m5py` - caveats

M5/M5’ use non-generalized tree concepts (e.g. « std » instead of « impurity ») – ‘translation’ is needed.

Pre-computing (weka’s « installing ») the smoothed models for speed at prediction time requires to reformulate the smoothing formula using linear model coefficients.

Some tree growing requirements are not available in sklearn’s `BaseDecisionTree`:
- Minimum impurity (std) for splitting = $5\% \times $ total

The Weka implementation slightly differs from the M5’ paper:
- RMSE is used to evaluate the models, instead of MAE
- new « Pruning factor » constant?
Not implemented, and new-but-not-in-the-paper

Not (yet) implemented
- The greedy linear model simplification process
- M5' enumeration & missing values handling

Benefits from using `sklearn` framework: the implementation can support any kind of models thanks to `sklearn`'s “clone”. However
- Not tested yet 😞
- Some features are disabled (e.g. smoothing can not be pre-installed)
m5py is an implementation of M5 / M5’ in python, available now from PyPi.

- It « unlocks » the pending scikit-learn#13732 so that the community can use it and polish it.
- Many improvements can still be done, do not hesitate to contribute if you are interested !

Yet, M5’ is just one of the many regression tree algorithms out there! For example:


2 python implementations: LightGBM and linear-tree
Towards Sustainable and Reproducible Results

How to make sure that our AI solutions have the appropriate level of quality?

Evaluation and benchmarking

- Reference datasets: private + public
- Datasets access tools: odsclient, cloudpathlib...
- Reproducible benchmarks: pytest, pytest-cases...
- Simulation of relearning strategies

“Open data” challenges

- 5 DRIVENDATA “power laws” competitions
- Microgrid energy management benchmark

Contributions to community efforts

- Scientific libraries: scikit-learn, pandas, etc.
- Tools: pytest, nox, requests, etc.
- Knowledge: stackoverflow, stackexchange...

Actionable documentation

- Reproducible doc: sphinx-gallery, mkdocs-gallery...
- API helpers azmlclient, pydantic-tables...
Open source - Highlights

Contributions to reference scientific libraries

- **scikit-learn**, improved $R^2$ and explained variance scores through a new `force_finite` parameter, 2022
- **pandas**, improved documentation for `to_datetime` – in particular about timezone handling, 2022
- **scikit-learn**, added a new randomized SVD solver to KernelPCA, 2021
- **scikit-learn**, fixed Kernel PCA numerical consistency between 32-64bits, 2020
- **scikit-learn**, statistical uniformity tests of the new `libsvm/liblinear` pseudo-random number generator, 2020
- **scikit-learn**, fixed pseudo-random number generator for `libsvm` and `liblinear`, 2020
- **scikit-learn**, added checks for eigenvalue decomposition numerical/conditioning issues in KernelPCA, 2019
- **scikit-learn**, fixed transform issue in KernelPCA when zero eigenvalues are present and not removed, 2019

Stackoverflow / StackExchange knowledge sharing

- Example users: Sylvain Marié, Schneider Electric | PyCon.DE & PyData 2022