Machine Learning for Micro-controllers

Technical Presentation

Antoine CARME (202? - WIP)
Antoine.CARME@outlook.com

https://github.com/antoinecarme/

Outline

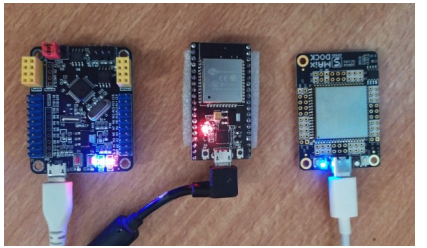
- History
- Goals
- Task Definition
- Prototyping

History

- Had some projects on industrializing machine learning models in various environments
  - Databases : generate SQL code from already trained scikit-learn/pytorch/keras/caret models.
    - Very standardized authoring, algebraic/tabular/columnar.
    - Almost unlimited resources,
    - Small variations between database providers. Need to manage a small set of exceptions for each database.
  - Micro-controllers : generate hardware-level C++ code from already trained models.
    - Very constrained
    - slow cpu,
    - power/watts : can run on battery.
    - Memory (< 256KB for some).
    - The generated C++ code should work for all platforms. Some platforms do not have numerical libraries. KISS.
- Only for model deployment and scoring : predict API calls.
- A lot of time reading/inspecting/ scikit-learn/caret/keras/pytorch code and other mathematical libraries (numpy. Scipy,) and pandas/SQL etc.
  - A lot of things in common. The same abstractions are finally used to generate SQL/C++ code for four different NN layers implementations (keras/pytorch/scikit-learn/caret).
  - Code reading in this case is very special !!!
  - Can be time-consuming. Limited time per model/feature. Define more priority for default settings and use cases.
- Software QA / Evaluation is based on the reference python version : SQL/C++ codes should give the same result (up to 12 decimal points) on a fixed set of models and datasets.
  - Process in place.
  - The same validation process for all models and databases.

Goals

- Provide some advanced machine-learning capabilities to the most frequent platforms.
  - ESP32 micro-controller on battery needs to analyze sensors data for months, build a decision tree model on the fly twice a day and send a weekly summary report as JSON.
  - A TinyML system.
- Do more than just predicting values from an existing model.
  - Training capabilities on small datasets (tens of columns).
    - Out-Of-Memory error is sometimes allowed.
  - Support the 4 most-used scikit-learn models.
    - Decision Tree (prototyping), Ridge, SVM, MLP
    - Classifications and regressions.
    - More models if need.
  - Allow building (fit), deploying (predict) and saving models.
  - In a portable, uniform, reliable and transparent way (next 4 slides).



Portable

- Portability : Good old C++.
  - Works as well on a huge Xeon server, a VPN Openwrt router, an Android mobile or a laptop.
  - GCC everywhere !!!
- Allows all possible data formats (dtypes) : INT8, FP16 etc.
  - Can be optimized to use less memory at no additional cost.
- Can be used in other languages through their respective CFFI
  - scikit-learn models for Common Lisp, Julia and R.
- Can be used in database as a UDF code.
- KISS. No use of external libraries.
- STL is allowed.
- Header only, No binary.
- Can be faster than scikit-learn and its python layers (cython, GIL) on the same hardware.
- Can be written/authored in a micro-controller friendly way:
  - Optimize for firmware size. Do it the embedded way !!!
  - Separate implementations for regressions and classifications. No need for AUC or GINI code when building a regression model.
  - More context-based analysis, depending on the model.

Uniform

- Uniform : Scikit-learn like API.
- Doc and training is not needed.
- Fit, predict, predict-proba.
- Can be used in existing reporting software : Jupyter notebooks, etc.
  - Python allows duck-typing.

Reliable

- Computations are validated against an existing software reference.
- Weights are similar to those given by scikit-learn or keras for the same model, dataset, settings.
-

Transparent

- The built model must be human-readable, auditable, debuggable.
- Text-based serialization.
- JSON import/export.
  - The model is built in a ESP32 and can be sent through the USB/Bluetooth interface, to be later loaded and analyzed in a more user-friendly environment.
- The implementation itself is transparent:
  - Reproducible. Source will be available under an open source license.
  - Header-only, no binary blobs.
  - Versioned.
  -

谢谢 !!!!