Snap ML: Optimized Machine Learning

Haris Pozidis, PhD
IBM Research - in collaboration with IBM Systems

https://www.zurich.ibm.com/snapml/

Snap ML: Optimizing Machine Learning

Design Objectives

- Accelerate & scale popular ML algorithms through system awareness, and HW/SW co-design
- Develop novel ML algorithms with best-in-class accuracy for business-focused applications

AI in Business – Challenges

Speed	Efficiency	Accuracy	Data Size
Train and re-train on new data online	Use resources judiciously	Make accurate decisions or predictions	Learn from all available data
Large parameter, model searches	Less resources means less $	Cost savings (e.g. card fraud), higher revenue (e.g. portfolio allocation)	More data, better models, higher accuracy
Make fast decisions	On-prem and in the cloud		Handle big data efficiently

↔

Snap ML is:

- Fast
- Scalable
- Accurate
- Resource-efficient
- Consumable

Snap ML Library Overview

WMLCE 1.6.0 (4Q18)

- Linear Regression
- Logistic Regression
- SVM

WMLCE 1.6.1 (2Q19)

- Decision Trees
- Random Forest

WMLCE 1.6.2 (4Q19)

- Boosting Machine

WMLCE 1.8.0 (3Q20)

- Random Forest Inference
- Boosting Machine Inference

1.7.0 (Feb’20) includes CPU/GPU-accelerated Decision Tree and multi-GPU Random Forest

1.8.0 (3Q20): CPU-Accelerated Inference for SnapBoost & new SnapBoost version (better accuracy)

Core library written in C++/CUDA

Python wrappers for APIs

~16k lines of code

Snap ML is integrated in IBM Watson Machine Learning Community Edition (WMLCE)

Seven library releases in WMLCE since Jun’18

WMLCE 1.7.0 (Feb’20)

Model	CPU solver	GPU solver	Multi-node
Linear Regression			
Logistic Regression			
SVM			
Decision Tree			
Random Forest			2H’20
Boosting Machine			2H’20

Key Innovations

Distributed Training

Adaptive Distributed Newton (**ADN**) algorithm for scaling out GLMs across a cluster of CPUs/GPUs

ICML 2018
NeurIPS 2018

Multi-threaded Training

State-of-the-art solvers on multi-core, multi-socket CPUs.

MLSys 2018
NeurIPS 2019

GPU Acceleration

Twice-parallel, asynchronous stochastic coordinate descent (**TPA-SCD**) for training linear models on GPUs.

Duality-gap based heterogenous learning (**DuHL**) for when the dataset does not fit in GPU memory.

NIPS 2017, FGCS 2018

Tree Ensembles

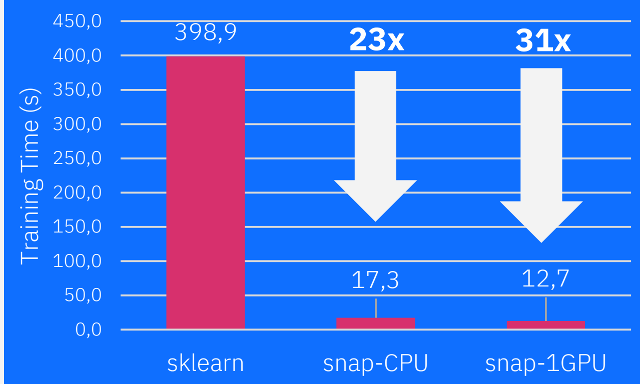
Memory-efficient breadth-first search algorithm for training of decision trees, random forests and gradient boosting machines.

New boosting algorithm based on stochastic selection of base learner.

MLSys 2019

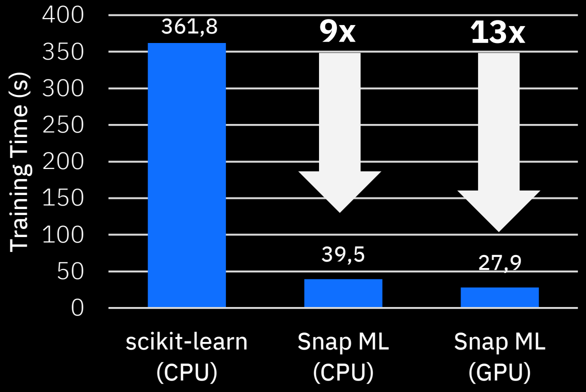
Fast: Seamless acceleration of scikit-learn applications

Logistic Regression (Kaggle #1 most used)



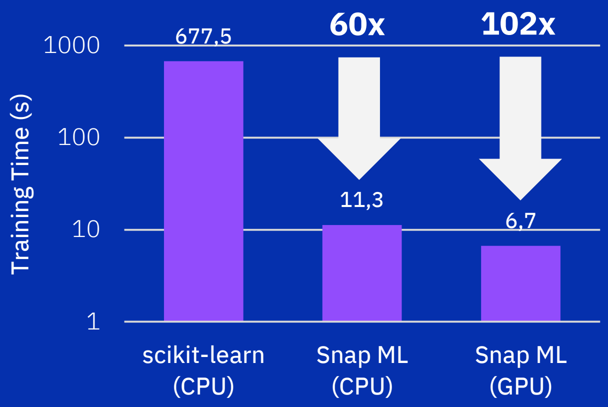
Dataset: criteo-kaggle
Machine: 1x AC922
scikit-learn loss: 0.4558
Snap ML loss: 0.4558

Random Forest (Kaggle #2 most used)



Dataset: SUSY
Machine: x86, V100
scikit-learn acc: 0.7848
Snap ML acc: 0.7945

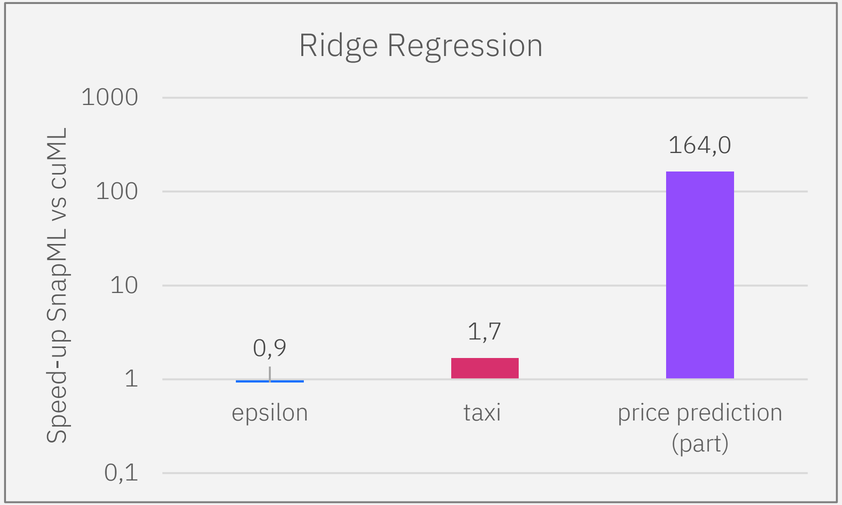
Decision Tree (Kaggle #2 most used)

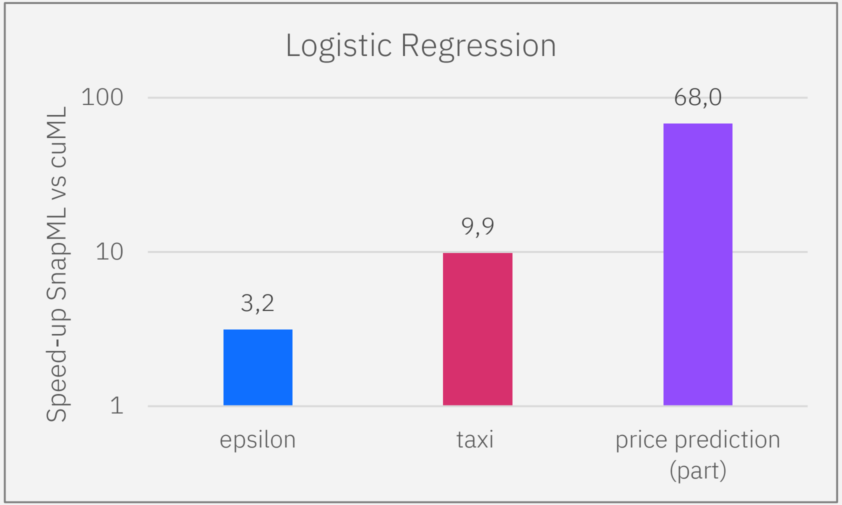


Dataset: higgs
Machine: x86, V100
scikit-learn acc: 0.7030
Snap ML acc: 0.7030

Snap ML vs. Nvidia RAPIDS (cuML)

SnapML speedup vs. RAPIDS/cuML: Ridge/Logistic Regression

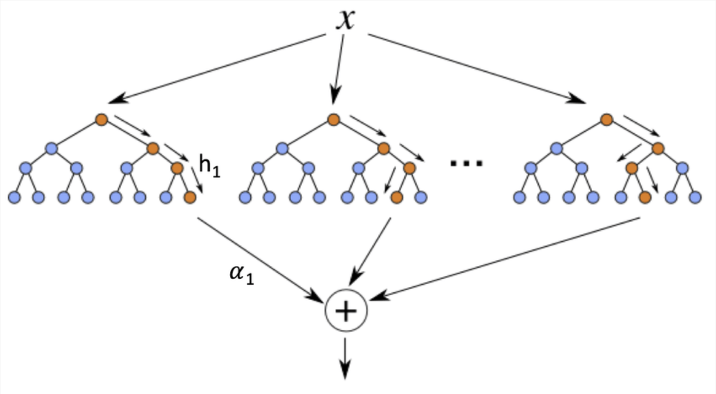




HW: Intel® Xeon® Gold 6150 CPU, 36 cores, 502 GB RAM, 1x Tesla V100 GPU, 16GB RAM
SW: Ubuntu 18.04.3, CUDA 10.1.243
Frameworks: cuml- 0.10.0, numpy- 1.17.3, scikit-learn- 0.21.3, WML-CE 1.6.2, pai4sk 1.5.0
Results are averages over best 5 out of 10 runs

Epsilon: 300,000 rows x 2,000 columns
Taxi: 1,600,000 rows x 606 columns
Price Pred.: 175,000 x 5074 columns

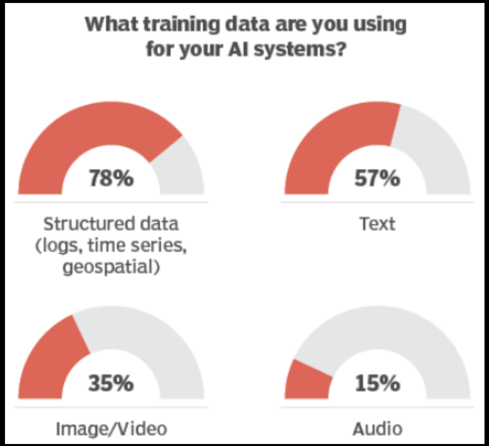
Gradient Boosting



GB models comprise an ensemble of decision trees, similar to a random forest (RF).

Deep neural networks achieve state-of-the-art accuracy on image, audio and NLP tasks.

However, on structured datasets GB usually out-performs all other models in terms of accuracy.



Majority of data in business is **Structured data**

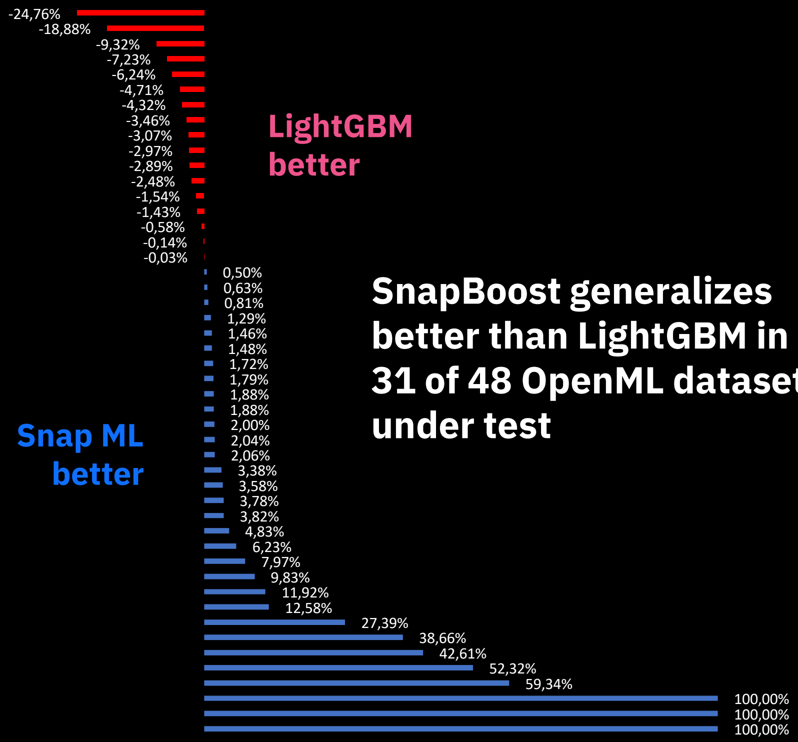
Source: The State of AI in the Enterprise, O'Reilly, 2019

Main GB libraries are XGBoost, LightGBM, and CatBoost.

Snap ML Gradient Boosting Machine (SnapBoost) targets **high accuracy** by a stochastic combination of base learners.

Accuracy vs. LightGBM

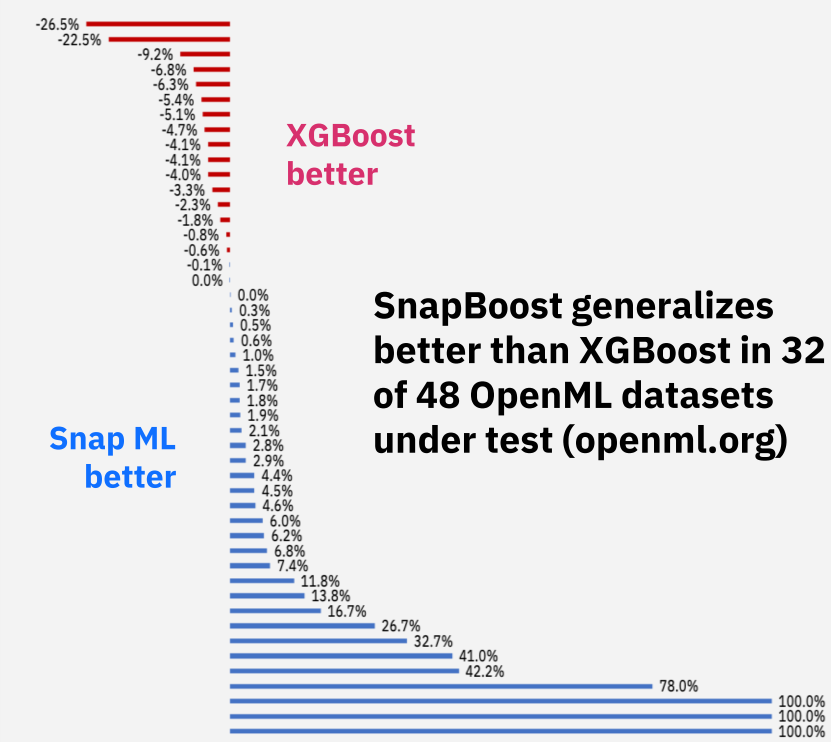
Relative Improvement in Test Loss (vs. LightGBM)



SnapBoost generalizes better than LightGBM in 31 of 48 OpenML datasets under test

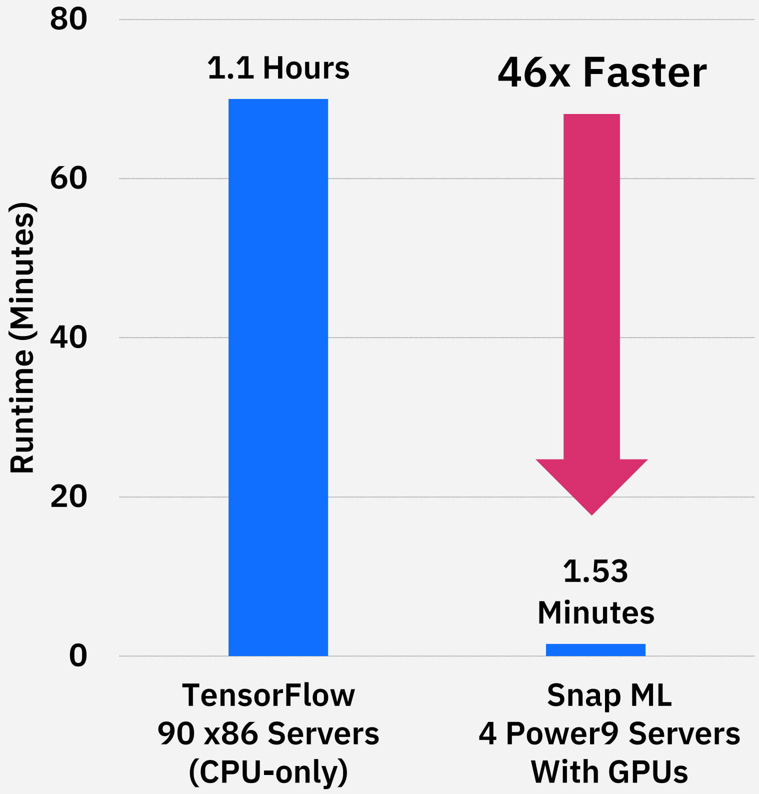
Accuracy vs. XGBoost

Relative Improvement in Test Loss (vs. XGBoost)



SnapBoost generalizes better than XGBoost in 32 of 48 OpenML datasets under test (openml.org)

Scalable: Handling Terabyte-scale Datasets



For huge data-sets that do not fit into CPU memory, scikit-learn cannot be used for training.

Instead, one must turn to distributed frameworks like TensorFlow or Spark MLlib

Snap ML applications can be seamlessly distributed across a cluster **without any code change**

Dataset: Criteo Terabyte Click Logs (http://labs.criteo.com/2013/12/download-terabyte-click-logs/)
4 billion training examples, 1 million features
Model: Logistic Regression: TensorFlow vs Snap ML
Test LogLoss: 0.1293 (Google using Tensorflow), 0.1292 (Snap ML)
Platform: 89 CPU-only machines in Google using Tensorflow versus 4 AC922 servers (each 2 Power9 CPUs + 4 V100 GPUs) for Snap ML
Google data from this Google blog

Consumable: RandomForest API

sklearn RandomForest API

```
from sklearn.ensemble import RandomForestClassifier
as SklearnForest

rf = SklearnForest(random_state=random_state,
                   max_depth=max_depth,
                   n_estimators=n_estimators,
                   n_jobs=num_threads,
                   max_features='sqrt')
```

```
# Train
rf.fit(X_train, y_train)

# Inference
pred_test = rf.predict(X_test)
```

SnapML RandomForest API

```
from snap_ml import RandomForestClassifier
as SnapForest

rf = SnapForest(random_state=random_state,
                max_depth=max_depth,
                n_estimators=n_estimators,
                n_jobs=num_threads,
                max_features='sqrt',
                use_histograms=True,
                use_gpu=True,
                gpu_ids=[0])
```

```
# Train
rf.fit(X_train, y_train)

# Inference
pred_test = rf.predict(X_test)
```

SnapBoost API

XGBoost sklearn API

```
from xgboost import XGBClassifier

booster = XGBClassifier(n_estimators=1000,
                        max_depth=8,
                        learning_rate=0.01
                        tree_method = 'gpu_hist',
                        n_jobs=8)

# Train
booster.fit(X_train, y_train)

# Inference
yhat_test = booster.predict(X_test, output_margin=True)
```

SnapBoost API

```
from snap_ml import BoostingMachine

booster = BoostingMachine(objective='logloss',
                          num_round=1000,
                          min_max_depth=8,
                          max_max_depth=8,
                          learning_rate=0.01,
                          use_gpu=True,
                          n_threads=8)

# Train
booster.fit(X_train, y_train)

# Inference
yhat_test = booster.predict(X_test)
```


Reuse additional functionalities of scikit-learn (pre-processing, filtering, feature engineering, train/test splitting)

Snap ML Evolution

1.5.2 06/18
- GLMs: Logistic regression, SVM, linear regression
- Local (single-node), MPI, Spark Python APIs
- Multi-GPU training, multi-threaded CPU training

1.5.3 09/18
- GLMs: sparse linear models
- Multiclass classification
- Visualization extensions

1.5.4 11/18
- CPU performance optimizations
- Fast (multi-threaded) inference
- Fast SVM Light data loader

1.6.0 03/19
- Decision Tree: CPU single-node
- Random Forest: CPU single-node
- cuDF integration for fast pre-processing

1.6.1 06/19
- Deep integration with Spark for Data Movement

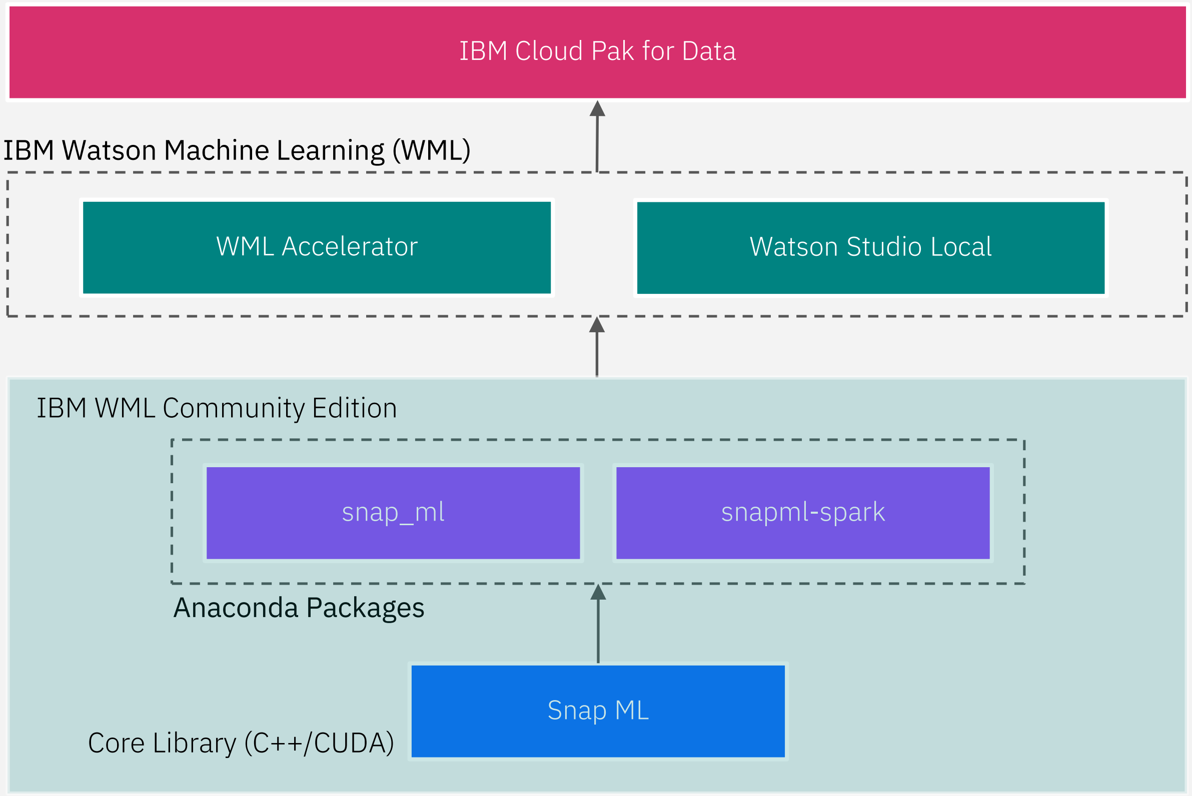
1.6.2 10/19
- Boosting Machine Tech Preview (CPU/GPU solvers)
- Sample weighting for imbalanced datasets (Decision Tree, Random Forest)

1.7.0 02/20
- Multithreaded tree-based models
- GPU-accelerated tree-based models
- Multi-GPU Random Forest

Roadmap

- Improved Boosting Machine (2Q20)
- Accelerated Inference for tree ensembles (2Q20)
- Distributed (multi-node) versions of Random Forest and Boosting Machine (2H20)
- Sparse data support for tree models (2H20)
- Spark integration of distributed tree-ensemble models (2021)

Snap ML Integration at IBM



Goal: Much broader reach and adoption through community channels

Where to get / How to try Snap ML

Available through IBM Watson ML Community Edition

- Free to download from
  https://developer.ibm.com/linuxonpower/deep-learning-powerai/releases/
- Runs in Power and x86 platforms
- Delivery through Conda packaging (“`conda install pai4sk`”)
- Documentation: https://ibmsoe.github.io/snap-ml-doc/index.html
- Examples: https://ibmsoe.github.io/snap-ml-doc/v1.6.0/tutorials.html
- Video: https://developer.ibm.com/videos/train-logistic-regression-and-random-forest-models-for-credit-default-prediction-using-snap-machine-learning/
- Blogs and articles:
  https://medium.com/@sumitg_16893/snap-ml-2x-faster-machine-learning-than-scikit-learn-c3529a1a6172
  https://www.ibm.com/blogs/systems/power-snapml-watson-machine-learning/
  https://developer.ibm.com/linuxonpower/2018/12/02/running-snapml-applications-with-ibm-powerai-enterprise-1-1-2/
  https://developer.ibm.com/series/snapml-on-powerai/
  https://developer.ibm.com/blogs/snap-ml-use-cases-blog/
  https://developer.ibm.com/linuxonpower/2020/03/26/benchmarking-linear-models-of-machine-learning-ml-frameworks-snap-ml-versus-cuml/