Flexible AutoML :
Accelerating AutoML adoption across Amazon

Abhishek Divekar (adivekar@amazon.com)
International Machine Learning, Amazon

Current approach to AutoML

- Popular AutoML systems available
  - *AutoGluon AWS AutoPilot, Google AutoML Tables, TPOT, H2O.ai, etc.*

- User Approach
  - AutoML user gathers training dataset.
  - AutoML system creates a model.

- Result
  - Case A: model meets bar on performance and latency, can be deployed
  - Case B: model does not meet performance bar, use other methods
  - Case C (common): model misses performance by 1%, or is too slow
  - Case D (common): data Scientist builds custom model, needs 6+ months of ML Engineer effort to be deployed

Persona 1: AutoML for Non-Tech users

Non-Tech users (Data associates, Product managers, etc) build model using AutoML system as ***black-box***

Issue - project is stuck if AutoML does not work in first try

Common Pitfall

Model misses performance by 1%, or is too slow (Case C)

Possible Solution

Allow user to open the black-box and customize a few parameters (suggested by a Scientist) that improves performance or latency

Requirement

Flexible AutoML system where any component can be customized

Persona 2: AutoML for Data Scientists

Data Scientists are good at using standard ML modeling best-practices* but face challenges while productionizing:

Select models looking at latency and code-dependencies in the deployment environment	Simulating the deployment environment takes extra engineering effort
Use K-fold Cross-Validation performance to select between modeling approaches	Training K+1 models is slow; Parallel setup is needed to make experimentation viable
Don't tune hyperparameters by hand, use Random Search / Bayesian Optimization / BOHB	Needs high number of resources. Non-trivial when also doing K-fold Cross-validation
Don't use XGBoost/BERT/ResNet as the only approach, but try many algorithms	SOTA approaches are often not easy to train or deploy, or only work on certain datasets.
Measure multiple metrics & tune hyperparameters based on business metrics	Metric-calculation code is complex and difficult inject into the tuning process

Common Pitfall

(Case D) Capable of building custom models but needs 6+ months of engineering effort to be deployed

Possible Solution

AutoML-augmented Data Science

- AutoML systems → robust code to train, tune and deploy models.
- Leverage 90% of AutoML system, customizes 10% → let the expert (Data Scientist) decide on a case-by-case basis

Requires

Flexible AutoML system where any component can be customized

[1] How to avoid Machine Learning Pitfalls, Michael A. Lones; [2] Machine Learning Yearning, Andrew Ng; [3] Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning, Sebastian Raschka

Litmus

A platform to offer flexible AutoML and accelerate AutoML adoption

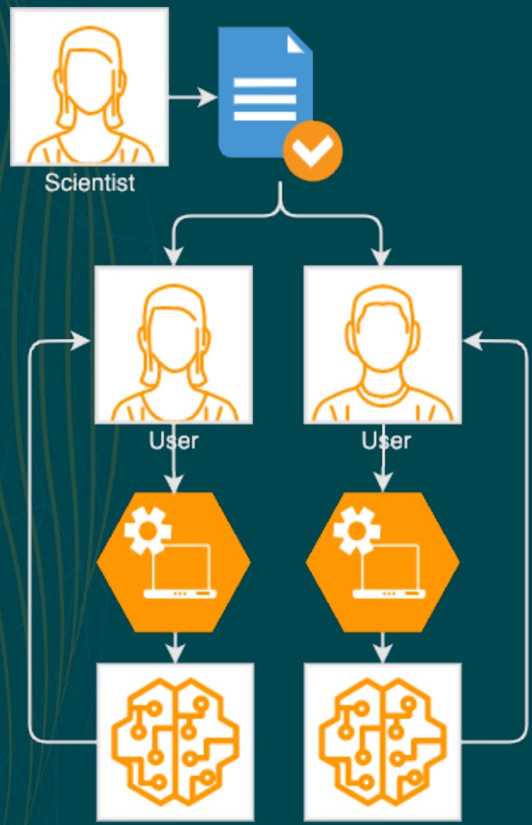
Flexible AutoML for non-tech users

“Standard” AutoML

Non-Tech users build model using AutoML system as black-box

Issue: project is stuck if AutoML does not work on first try.

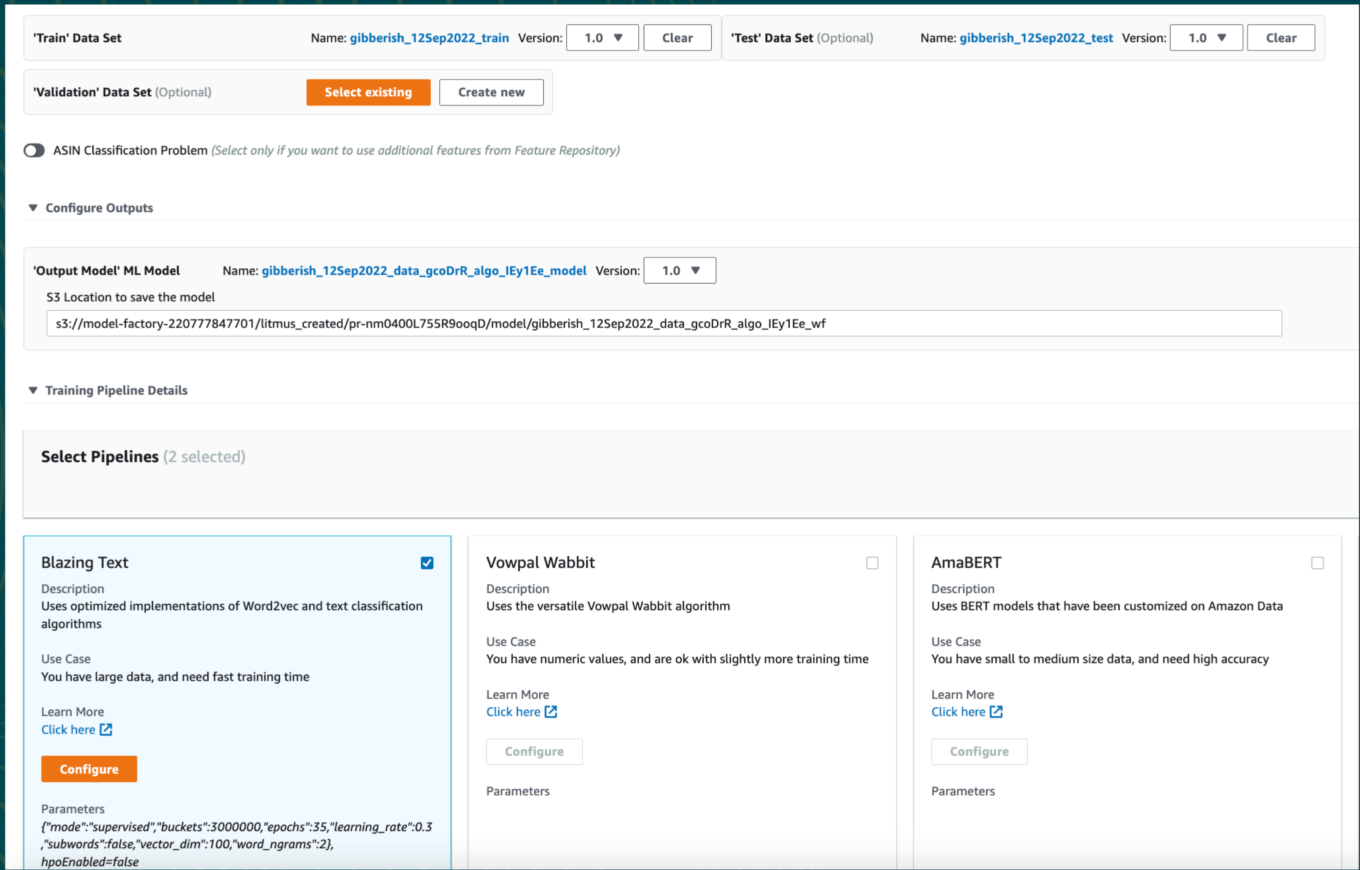
“Flexible” AutoML



1. Data Scientist specifies AutoML ***recipe***:
   - ➢ Try pipeline-A/B/C
   - ➢ Gather data for class X
2. Non-tech users iterate through recipe until model meets performance bar
3. Reach out to Data Scientist for possible customization if recipe does not work

Advantage: Operations teams can build models with minimal Scientist support

Simple UX for non-tech users



Simple UX for training

- User selects model based on recipe, else we select based on data
- Automatic hyperparameter tuning, K-fold, etc

Result

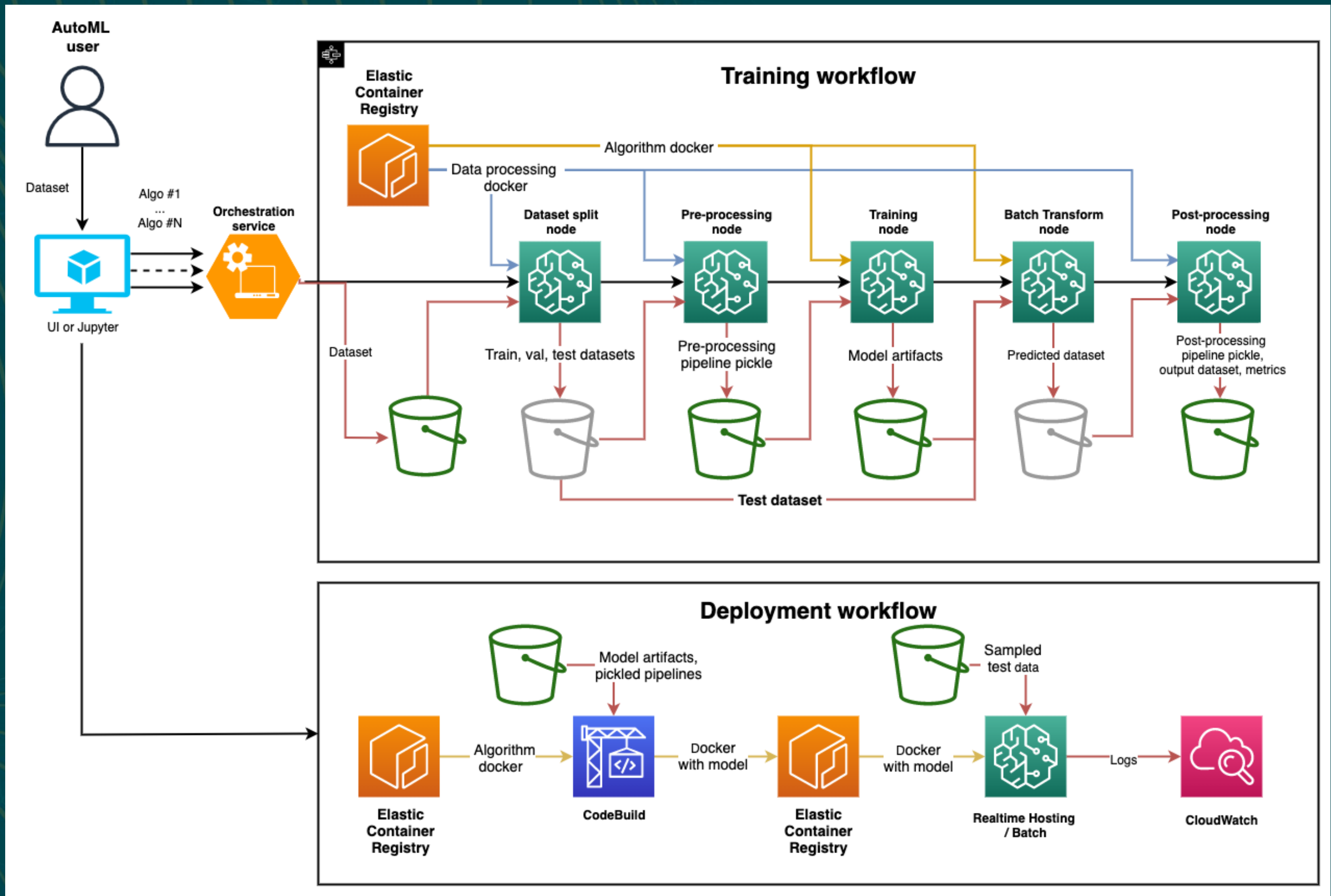
- Used by team of data associates to deploy 500+ models on Amazon website

Configurable UX for Scientists

```
litmus.train(
    data=[
        TaskData.create(
            name='gl_classification_data',
            task=('multi-class', 'classification'),
            data={'train': 's3://...', 'test': 's3://...', },
            schema={
                ## ...
            },
            k_fold=5 ## Or, a KFoldCV object
        )
    ],
    algorithm=[
        'XGBoost',  ## Creates litmus.XGBoost() with default hyperparams
        litmus.XGBoost().hyperparams(max_depth=5),
        CustomAlgorithm().hyperparams()
    ],
    metrics={
        'train': ['accuracy'],
        'validation': ['accuracy'],
        'test': ['accuracy', Metric('coverage_at_precision', {'precision'=0.85})]
    },
    resources={
        'train': {'gpus':1},
        'predict': {'gpus':6},
    }
)
```

- Easy dataset, hyperparameter configuration, K-fold, etc.
- Distributed training: simply set “gpus=8”
- Detailed metrics
- Anything can be customized:
  - Pre-processing logic
  - Post-processing logic
  - Algorithm code
  - Metrics

Unified backend system



Benefits:

1) Any trained model is 1-click deployable
2) Scalable
3) Low infrastructure maintenance

→ control flow
→ data flow
→ pull Dockers
→ push Dockers to container registry

Issue: Scaling data-processing code

- Pandas: extremely popular
  - 34k+ companies using Pandas in 2022 [1]
  - Simple, flexible API for prototyping/analysis
  - Decent speed with small resources (1 CPU)
- When deploying, Pandas is slow:
  - Text-preprocessing pipeline:
    - 28 ms for 1 row.
    - ~9 minutes for 10MM rows
  - Low-latency use cases:
    - Chatbot responses: <50ms latency
    - Ads recommendation: <5ms latency
    - Slow data-processing restricts complexity of ML models.

[1] https://discovery.hgdata.com/product/pandas

Solutions?

- Modin / Dask / Spark:
  - Drop-in replacement for Pandas
  - Slower than Pandas for small data
- NumPy + Numba / Dict processing:
  - Fast (20-50x faster than Pandas)
  - Loses simplicity of Pandas API
  - Bugs can be introduced when translating code from Pandas

Litmus Scalable DataFrame (LitSDF)

Idea:

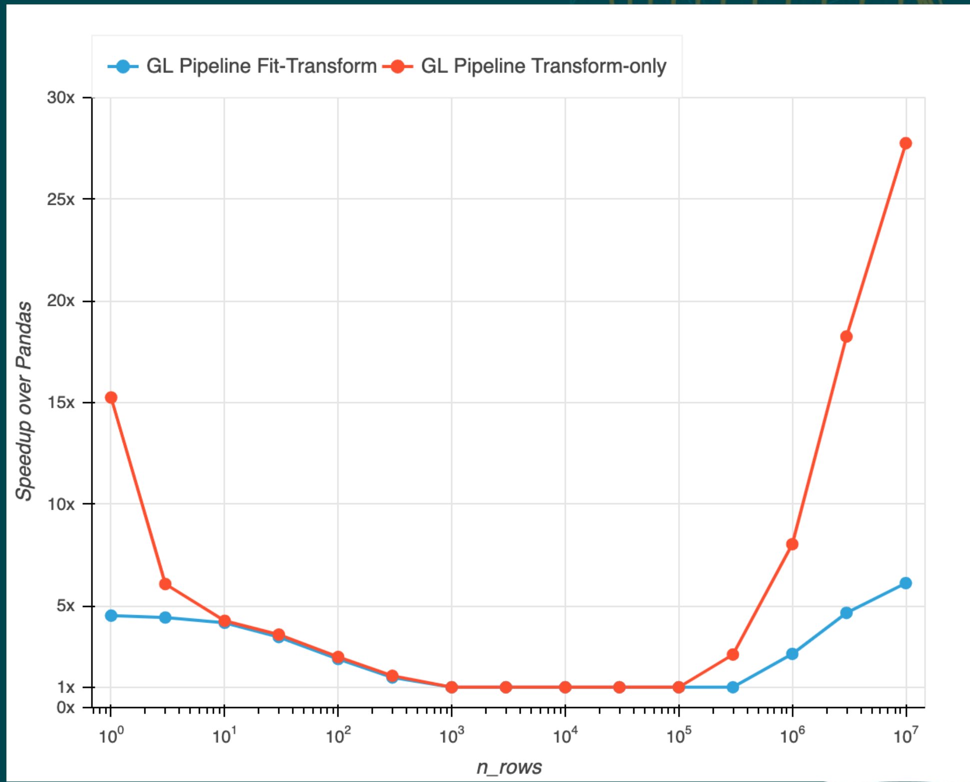
- Expose the Pandas API, but implement different data-layouts under the hood
- Scientists/Engineers can write code in Pandas, but it runs using Numpy / Dicts / Modin / Dask etc. Might use Pandas itself.
- During deployment, select optimal layout:
  - Static: based on number of incoming rows
  - Dynamic: use a bandit algorithm / Reinforcement Learning
- Can support upcoming dataframe layouts:
  - Vaex (memory-mapped dataframe)
  - cuDF (GPU dataframe)

LitSDF: Speedup over Pandas

- Training:
  - 4.5x faster for 1 row (Dict)
  - Use Pandas for 1k-100k rows
  - 6.1x faster for 10MM rows (Dask)

- Data processing (post deployment):
  - 15.2x faster for 1 row (Dict)
  - Use Pandas for 1k-100k rows
  - 27.7x faster for 10MM rows (Dask)

❖ LitSDF is a general-purpose library, can be used during experimentation, ETL jobs, etc.



Acknowledgements

Litmus team:

- Abhishek Divekar
- Gaurav Manchanda
- Siba Rajendran
- Ramakrishna Nalam
- Mohit Gupta
- Vivek Sembium

International Machine Learning, Amazon

Thank You

Abhishek Divekar (adivekar@amazon.com)
International Machine Learning, Amazon