Automated Machine Learning (AutoML) in Insurance

Panyi Dong

University of Illinois Urbana-Champaign

2025-02-07

Synopsis

- **Why** we need AutoML
- **What** is our AutoML
- **How** our AutoML performs

Introduction: ML

- Machine Learning (ML) "**leverages data** to improve performance on some set of tasks”.
- Real-life application scenarios :
    - Self-driving cars
    - Recommendation systems
    - Automated translation
    - ...
- For insurance industry:
    - Future claim estimation
    - Fraud detection
    - Automated underwriting
    - ...

Introduction: ML

- However, ML tasks can be
  - **Experience-dependent**
  - **Heavy manual work**
- By the **data-driven** nature of ML algorithms, selection of models and hyperparameters are critical, and **no universal solution** exists.
- Furthermore, industrial datasets add to the complexity
  - Not well-formatted or well-organized datasets
    - **Missing values**
    - **Irrelevant features**
    - **Imbalance distributions**
- It's difficult for those who have no previous experience/knowledge to gain hands-on experience.

Introduction: AutoML

- Automated Machine Learning (AutoML) is one of the solutions.
  - AutoML tries to
    - Automatically select a ML model
    - Automatically tune for optimal hyperparameters
  - “non-expert users” can apply ML to their application scenarios more effectively

Introduction: AutoML

- A ML model M can be characterized by
  - Parameters θ
  - Hyperparameters λ
- The learning can be formulated as

$$\underset{\theta}{\operatorname{argmin}} \mathcal{L}(M_{\lambda}^{\theta}(\mathbf{X}), \mathbf{y})$$

  - dataset $\mathcal{D} = (\mathbf{X}, \mathbf{y})$
  - loss function $\mathcal{L}$
- The choice of model M and hyperparameter set λ is important

Introduction: AutoML

- Model selection finds optimal model architecture M^*

$$M^* = \underset{M \in \mathcal{M}}{\operatorname{argmin}} \mathbb{E}_{\mathcal{D} \sim (\mathcal{D}_{train}, \mathcal{D}_{valid})} \mathcal{V}(\mathcal{L}, M_{\lambda_0}, \mathcal{D})$$

- Hyperparameter Optimization (HPO) finds optimal hyperparameter set λ^{M*}

$$\lambda^{M*} = \underset{\lambda^M \in \Lambda^M}{\operatorname{argmin}} \mathbb{E}_{\mathcal{D} \sim (\mathcal{D}_{train}, \mathcal{D}_{valid})} \mathcal{V}(\mathcal{L}, M_{\lambda^M}, \mathcal{D})$$

- Objective function $\mathcal{V}$
  - trains on train set $\mathcal{D}_{train}$
  - returns evaluation loss on valid set $\mathcal{D}_{valid}$

Introduction: AutoML

- To combine, one option is two-step process
- Another is Combined Algorithm Selection and Hyperparameter optimization (CASH)

$$
\begin{aligned}
M^*_{\lambda^*} &= \underset{M \in \mathcal{M}, \lambda^M \in \Lambda^M}{\operatorname{argmin}} \mathbb{E}_{\mathcal{D} \sim (\mathcal{D}_{train}, \mathcal{D}_{valid})} \mathcal{V}(\mathcal{L}, M_{\lambda^M}, \mathcal{D}) \\
&= \underset{(M, \lambda^M) \in \mathcal{C}^M}{\operatorname{argmin}} \mathbb{E}_{\mathcal{D} \sim (\mathcal{D}_{train}, \mathcal{D}_{valid})} \mathcal{V}(\mathcal{L}, M_{\lambda^M}, \mathcal{D})
\end{aligned}
$$

Introduction: AutoML

Our AutoML[1]

- Complete, fully functional processing and model tuning
- Special treatment for Insurance imbalanced datasets
  - Data Balancing
  - Pipeline ensemble
- Record training process and store the optimal pipeline for continued applications

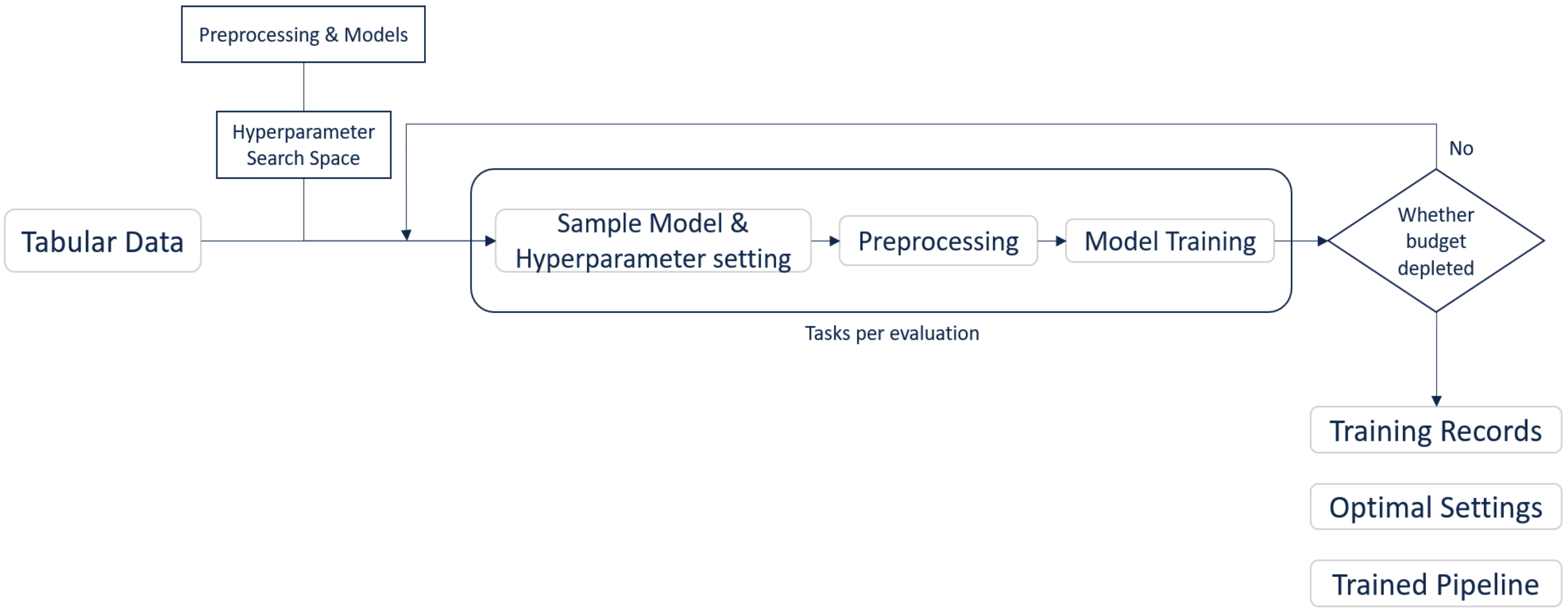
[1] https://github.com/PanyiDong/InsurAutoML

AutoML: Components

1. Data Encoding
2. Data Imputation
3. Data Balancing
4. Data Scaling
5. Feature Selection
6. Classification/Regression Models
7. Model Selection and Hyperparameter Optimization

Preprocessing (items 1–5)

AutoML: Workflow



AutoML: Optimization

Algorithm 1: The AutoML optimization

Input: Dataset $\mathcal{D} = (\mathcal{D}_{train}, \mathcal{D}_{valid})$; Search space $\mathcal{U}$; Time budget T; Evaluation budget G; Search algorithm $Samp$

Output: Optimal pipline with hyperparameter settings $\mathcal{P}^*$

$k = 0$; /* Round of evaluation */

$t^{re} = T$; /* Remaining time budget */

$g^{re} = G$; /* Remaining evaluation budget */

while $t^{re} > 0$ *and* $g^{re} > 0$ **do**

  $t^{start} = CurrentTime;$

  $(E^{(k)}, \lambda_E^{(k)}), (I^{(k)}, \lambda_I^{(k)}), (B^{(k)}, \lambda_B^{(k)}), (S^{(k)}, \lambda_S^{(k)}), (F^{(k)}, \lambda_F^{(k)}), (M^{(k)}, \lambda_M^{(k)}) = Samp^{(k)}(\mathcal{U});$

  $\mathcal{P}_k = M^{(k)}_{\lambda_M^{(k)}} \circ F^{(k)}_{\lambda_F^{(k)}} \circ S^{(k)}_{\lambda_S^{(k)}} \circ B^{(k)}_{\lambda_B^{(k)}} \circ I^{(k)}_{\lambda_I^{(k)}} \circ E^{(k)}_{\lambda_E^{(k)}};$

  $L^{eval,(k)} = \mathcal{V}(\mathcal{L}, \mathcal{P}_k, \mathcal{D});$

  $t^{end} = CurrentTime;$

  $k = k + 1;$

  $t^{re} = t^{re} - (t^{end} - t^{start});$

  $g^{re} = g^{re} - 1;$

end

$k^* = \underset{k}{\operatorname{argmin}} L^{eval,(k)}$; /* Find optimal pipeline order */

$\mathcal{P}^* = \mathcal{P}_{k^*};$

return $\mathcal{P}^*$;

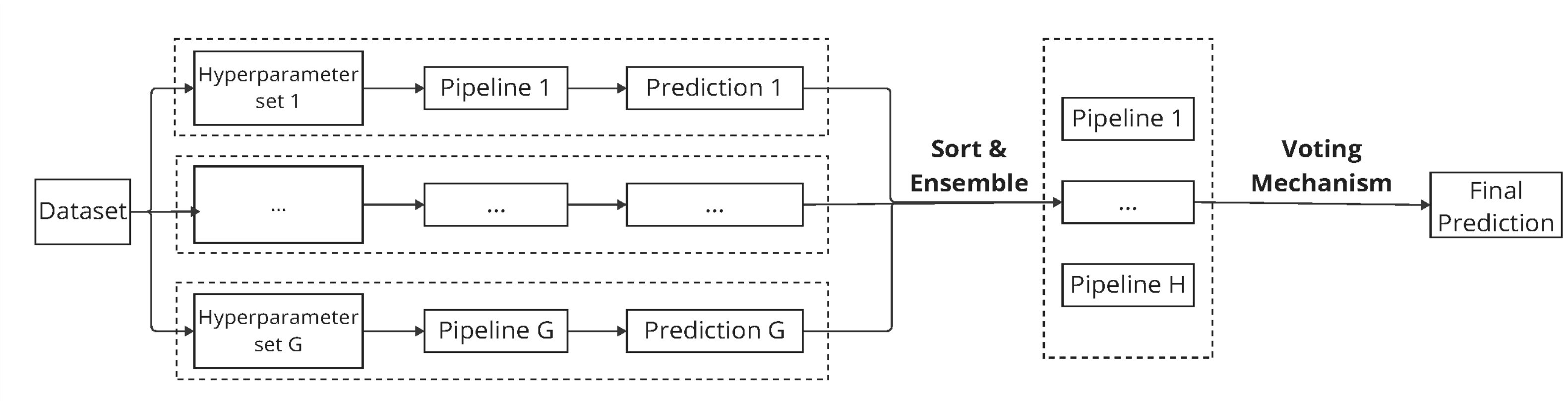
AutoML: Optimization

- To connect all components, we use **ray.tune** for model selection and hyperparameter optimization.
- **ray.tune** is a scalable Python package
  - to conduct experiments on hyperparameter tuning
  - compatible with common ML model structures
    - scikit-learn, TensorFlow, PyTorch, ...
  - compatible with search algorithms like
    - Optuna, HyperOpt, ...

AutoML: Ensemble

- Model Ensemble is common solution to
  - Data imbalance
  - State-of-the-art performance
- We adopt three ensemble structures
  - Stacking
    - Fully parallel training on whole set
  - Bagging
    - Parallel training on subsets
  - Boosting
    - Sequential training on residuals

AutoML: Stacking Ensemble



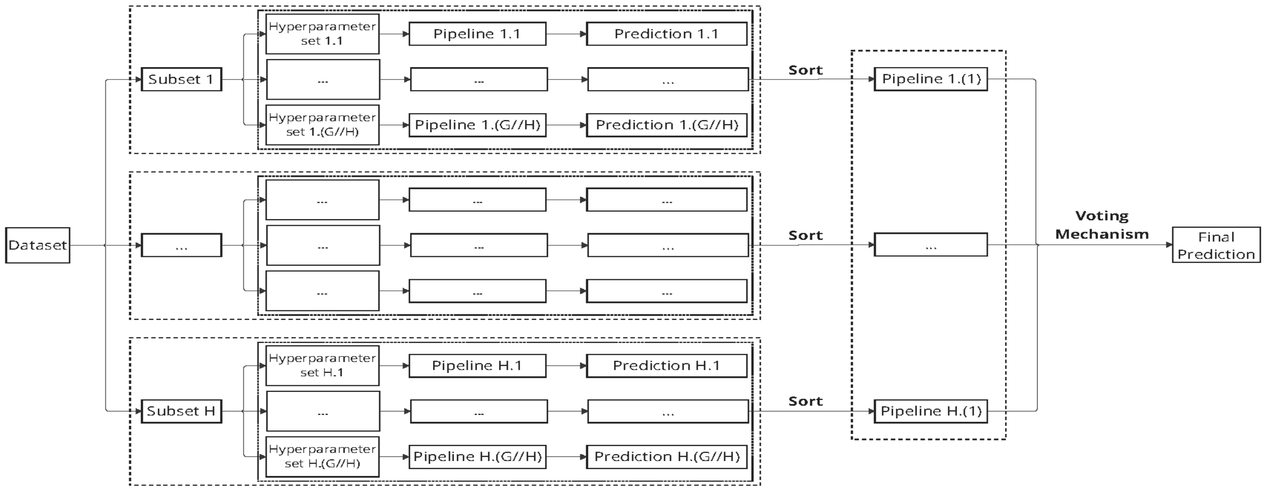
Algorithm 2: The Stacking Ensemble

Input: Dataset $\mathcal{D} = (\mathcal{D}_{train}, \mathcal{D}_{valid})$; Search space $\mathcal{U}$; Time budget T; Evaluation budget G; Search algorithm $Samp$; Size of the ensemble H

Output: Ensemble Σ_H

$k = 0$; `/* Round of evaluation */`
$t^{re} = T$; `/* Remaining time budget */`
$g^{re} = G$; `/* Remaining evaluation budget */`
while $t^{re} > 0$ *and* $g^{re} > 0$ **do**
  $t^{start} = CurrentTime$;
  $(E^{(k)}, \lambda_E^{(k)}), (I^{(k)}, \lambda_I^{(k)}), (B^{(k)}, \lambda_B^{(k)}), (S^{(k)}, \lambda_S^{(k)}), (F^{(k)}, \lambda_F^{(k)}), (M^{(k)}, \lambda_M^{(k)}) = Samp^{(k)}(\mathcal{U})$;
  $\mathcal{P}_k = M^{(k)}_{\lambda_M^{(k)}} \circ F^{(k)}_{\lambda_F^{(k)}} \circ S^{(k)}_{\lambda_S^{(k)}} \circ B^{(k)}_{\lambda_B^{(k)}} \circ I^{(k)}_{\lambda_I^{(k)}} \circ E^{(k)}_{\lambda_E^{(k)}}$;
  $L^{eval,(k)} = \mathcal{V}(\mathcal{L}, \mathcal{P}_k, \mathcal{D})$;
  $t^{end} = CurrentTime$;
  $k = k + 1$;
  $t^{re} = t^{re} - (t^{end} - t^{start})$;
  $g^{re} = g^{re} - 1$;
end
$\{\mathcal{P}_{(k)}\} = sort(\{\mathcal{P}_k\})$;
$\Sigma_H = \Sigma_H(\mathcal{P}_{(1)}, \mathcal{P}_{(2)}, \ldots, \mathcal{P}_{(H)})$;
return Σ_H;

AutoML: Bagging Ensemble



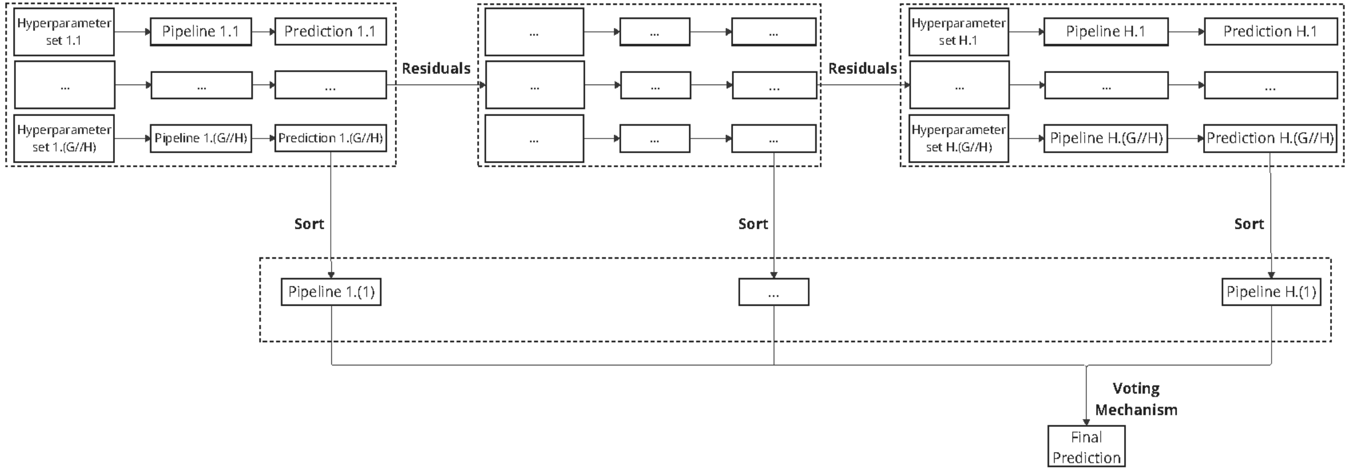
Algorithm 3: The Bagging Ensemble

Input: Dataset $\mathcal{D} = (\mathcal{D}_{train}, \mathcal{D}_{valid})$; Search space $\mathcal{U}$; Time budget T; Evaluation budget G; Search algorithm $Samp$; Size of the ensemble H; Subset Matrices $\{\boldsymbol{\rho}^{(h)}\}_{h=1,2,\ldots,H}$

Output: Ensemble Σ_H

for $h \leftarrow 1$ *to* H **do**
    $k = 0$; `/* Round of evaluation */`
    $t^{re} = T//H$; `/* Remaining time budget */`
    $g^{re} = G//H$; `/* Remaining evaluation budget */`
    $\mathcal{D}^{(h)} = ((\mathbf{X}_{train}\boldsymbol{\rho}^{(h)}, \mathbf{y}_{train}), (\mathbf{X}_{valid}\boldsymbol{\rho}^{(h)}, \mathbf{y}_{valid}))$;
    while $t^{re} > 0$ *and* $g^{re} > 0$ **do**
        $t^{start} = CurrentTime$;
        $(E^{(k)}, \lambda_E^{(k)}), (I^{(k)}, \lambda_I^{(k)}), (B^{(k)}, \lambda_B^{(k)}), (S^{(k)}, \lambda_S^{(k)}), (F^{(k)}, \lambda_F^{(k)}), (M^{(k)}, \lambda_M^{(k)}) = Samp^{(k)}(\mathcal{U})$;
        $\mathcal{P}_{h,k} = M^{(k)}_{\lambda_M^{(k)}} \circ F^{(k)}_{\lambda_F^{(k)}} \circ S^{(k)}_{\lambda_S^{(k)}} \circ B^{(k)}_{\lambda_B^{(k)}} \circ I^{(k)}_{\lambda_I^{(k)}} \circ E^{(k)}_{\lambda_E^{(k)}}$;
        $L^{eval,(k)} = \mathcal{V}(\mathcal{L}, \mathcal{P}_{h,k}, \mathcal{D}^{(h)})$;
        $t^{end} = CurrentTime$;
        $k = k + 1$;
        $t^{re} = t^{re} - (t^{end} - t^{start})$;
        $g^{re} = g^{re} - 1$;
    end
    $\{\mathcal{P}_{h,(k)}\} = sort(\{\mathcal{P}_{h,k}\})$;
end
$\Sigma_H = \Sigma_H(\mathcal{P}_{1,(1)}, \mathcal{P}_{2,(1)}, \ldots, \mathcal{P}_{H,(1)})$;
return Σ_H;

AutoML: Boosting Ensemble



Algorithm 4: The Boosting Ensemble

Input: Dataset $\mathcal{D} = (\mathcal{D}_{train}, \mathcal{D}_{valid})$; Search space $\mathcal{U}$; Time budget T; Evaluation budget G; Search algorithm $Samp$; Size of the ensemble H

Output: Ensemble Σ_M

Initialization: $\mathbf{y}_{train,0} = \mathbf{y}_{train}$; $\mathbf{y}_{valid,0} = \mathbf{y}_{valid}$; $\hat{\mathbf{y}}_{train,0} = 0$; $\hat{\mathbf{y}}_{valid,0} = 0$;

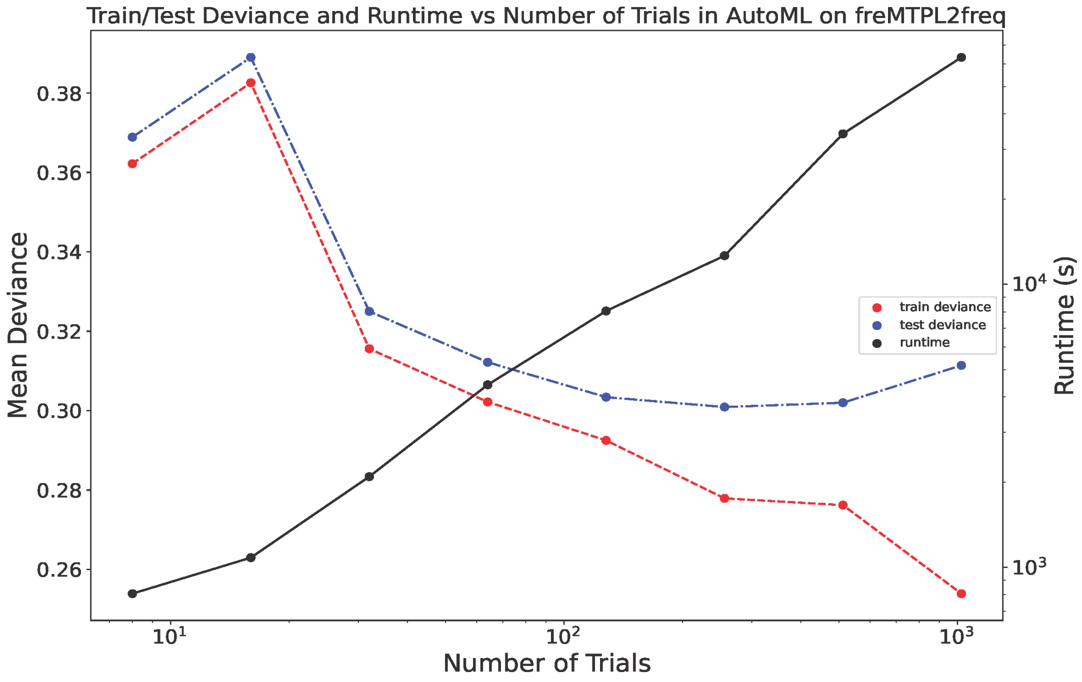
for $h \leftarrow 1$ *to* H **do**
- $k = 0$; /* Round of evaluation */
- $t^{re} = T//H$; /* Remaining time budget */
- $g^{re} = G//H$; /* Remaining evaluation budget */
- $\mathbf{y}_{train,h} = \mathbf{y}_{train,h-1} - \hat{\mathbf{y}}_{train,h-1}$; $\mathbf{y}_{valid,h} = \mathbf{y}_{valid,h-1} - \hat{\mathbf{y}}_{valid,h-1}$;
- $\mathcal{D}_h = ((\mathbf{X}_{train}, \mathbf{y}_{train,h}), (\mathbf{X}_{valid}, \mathbf{y}_{valid,h}))$;
- **while** $t^{re} > 0$ *and* $g^{re} > 0$ **do**
  - $t^{start} = CurrentTime$;
  - $(E^{(k)}, \lambda_E^{(k)}), (I^{(k)}, \lambda_I^{(k)}), (B^{(k)}, \lambda_B^{(k)}), (S^{(k)}, \lambda_S^{(k)}), (F^{(k)}, \lambda_F^{(k)}), (M^{(k)}, \lambda_M^{(k)}) = Samp^{(k)}(\mathcal{U})$;
  - $\mathcal{P}_{h,k} = M^{(k)}_{\lambda_M^{(k)}} \circ F^{(k)}_{\lambda_F^{(k)}} \circ S^{(k)}_{\lambda_S^{(k)}} \circ B^{(k)}_{\lambda_B^{(k)}} \circ I^{(k)}_{\lambda_I^{(k)}} \circ E^{(k)}_{\lambda_E^{(k)}}$;
  - $L^{eval,(k)} = \mathcal{V}(\mathcal{L}, \mathcal{P}_{h,k}, \mathcal{D}_h)$;
  - $t^{end} = CurrentTime$;
  - $k = k + 1$;
  - $t^{re} = t^{re} - (t^{end} - t^{start})$;
  - $g^{re} = g^{re} - 1$;
- **end**
- $\{\mathcal{P}_{h,(k)}\} = sort(\{\mathcal{P}_{h,k}\})$;
- $\hat{\mathbf{y}}_{train,h} = \mathcal{P}_{h,(1)}(\mathbf{X}_{train})$; $\hat{\mathbf{y}}_{valid,h} = \mathcal{P}_{h,(1)}(\mathbf{X}_{valid})$;

end

$\Sigma_H = \Sigma_H(\mathcal{P}_{1,(1)}, \mathcal{P}_{2,(1)}, \ldots, \mathcal{P}_{H,(1)})$;

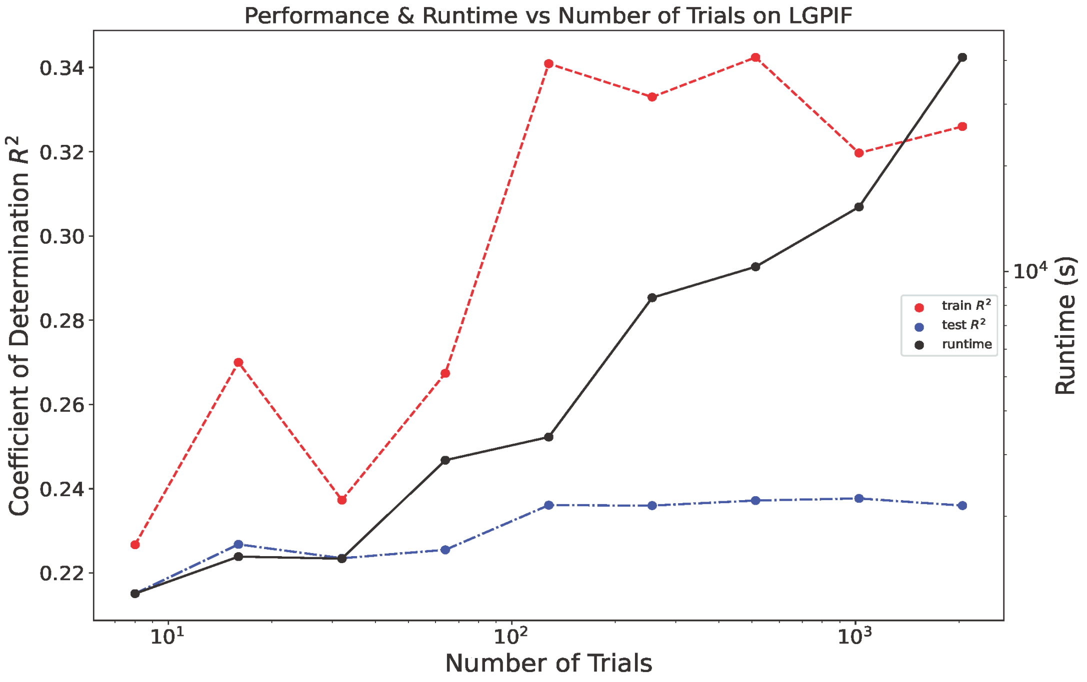
return Σ_H;

Experiments: French Motor Third-Part Liability



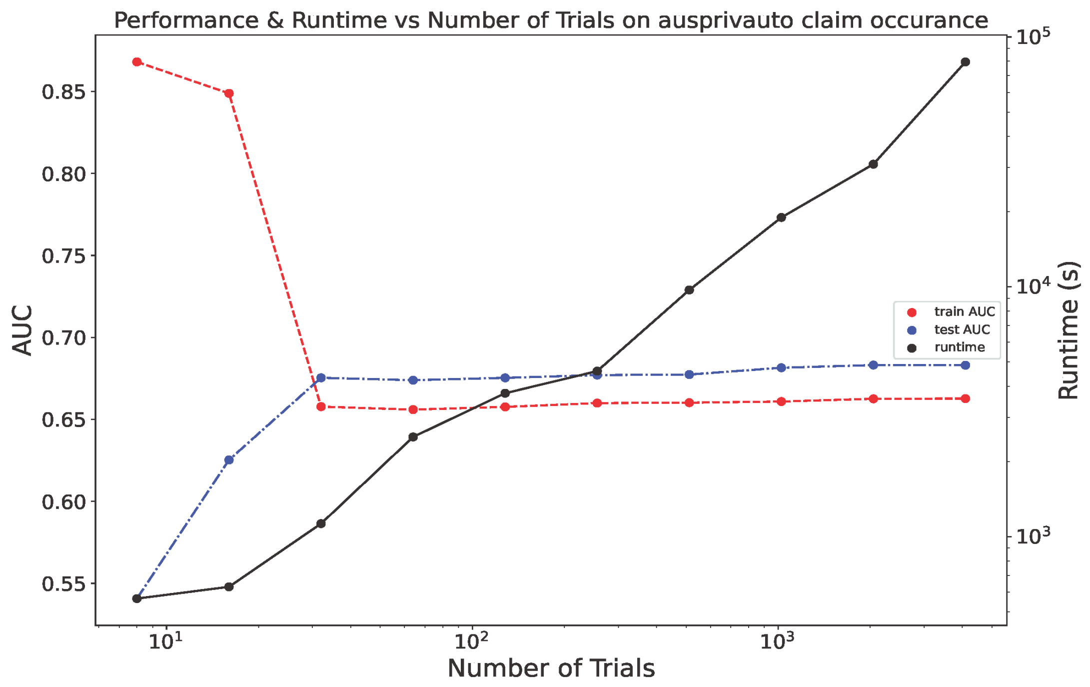
G	T/s	runtime/s	Train Deviance	Test Deviance
8	900	807.62	0.3622	0.3689
16	1,800	1,082.21	0.3826	0.3890
32	3,600	2,092.15	0.3156	0.3250
64	7,200	4,417.51	0.3022	0.3122
128	14,400	8,052.91	0.2925	0.3034
256	28,800	12,624.60	0.2779	**0.3009**
512	57,600	34,036.03	0.2762	0.3020
1024	115,200	63,401.81	**0.2539**	0.3114

Experiments: Wisconsin Local Government Property Insurance Fund



G	T/s	runtime/s	Train R^2	Test R^2
8	900	1202.55	0.2267	0.2151
16	1,800	1,533.87	0.2700	0.2268
32	3,600	1,513.57	0.2373	0.2235
64	7,200	2,891.36	0.2674	0.2255
128	14,400	3,367.43	0.3409	0.2361
256	28,800	8,413.55	0.3330	0.2360
512	57,600	10,313.03	**0.3424**	0.2372
1024	115,200	15,282.33	0.3197	**0.2377**
2048	230,400	40,856.35	0.3260	0.2360

Experiments: Automobile claim datasets in Australia



G	T/s	runtime/s	Train AUC	Test AUC
8	900	565.15	0.8681	0.5407
16	1,800	629.83	0.8489	0.6253
32	3,600	1,127.55	0.6578	0.6754
64	7,200	2,506.17	0.6560	0.6739
128	14,400	3,749.68	0.6576	0.6754
256	28,800	4,598.22	0.6600	0.6770
512	57,600	9,709.30	0.6602	0.6774
1024	115,200	18,938.53	0.6609	0.6815
2048	230,400	30,951.82	0.6626	**0.6831**
4096	460,800	79,438.58	**0.6627**	0.6831

Experiments: Comparison

Data		GLM Results		AutoML	
	Family	Metric	Test loss	G	Test loss
freMTPL2freq	Poisson	Poisson Deviance	0.3595	256	**0.3009**
LGPIF	Tweedie	R^2	0.2062	1024	**0.2377**
		Gini	0.4089		**0.4187**
		ME	0.1609		**0.0476**
		MSE	14.0533		**13.4956**
		MAE	**2.8749**		2.8955
ausprivauto_occ	Bernoulli	AUC	0.6792	2048	**0.6831**
ausprivauto_fre	Poisson	Poisson Deviance	0.4437	256	**0.3668**
ausprivauto	Tweedie	RMSE	1,091.6741	1024	**1,091.5361**

Experiments: Comparison

Data		Actuarial literature		AutoML	
	Source	Metric	Test loss	G	Test loss
freMTPL2freq	[2]	Poisson Deviance	0.3149	256	**0.3009**
LGPIF	[3]	R^2	0.229	1024	**0.2377**
		Gini	0.414		**0.4187**
		ME	0.048		**0.0476**
		MSE	13.651		**13.4956**
		MAE	**2.883**		2.8955
ausprivauto_occ	[4]	AUC	0.660	2048	**0.6831**

[2]Wuthrich, M. V. (2019). From generalized linear models to neural networks, and back. Technical report, Department of Mathematics, ETH Zurich.

[3] Quan, Z. and Valdez, E. A. (2018). Predictive analytics of insurance claims using multivariate decision trees. *Dependence Modeling*, 6(1):377–407.

[4] Si, J., He, H., Zhang, J., and Cao, X. (2022a). Automobile insurance claim occurrence prediction model based on ensemble learning. *Applied Stochastic Models in Business and Industry*, 38(6):1099–1112.

Conclusion

- Provide a workable pipeline
    - With focus on insurance imbalanced datasets
- Acceptable performance and efficiency
- Flexible framework for modification
- Provides prototype and insights for further improvement

Thank you! Q&A