ML 2025 Project

Authors :Truong Manh Nguyen, Thi Hien Do

Master degree in Data Science and Informatics.

Email: m.nguyen2@studenti.unipi.it, t.do1@studenti.unipi.it

Date 23/01/2005

Type of project: **B**

Objectives

Perform an extensive experimental comparison of different Machine Learning models using existing ML/NN tools.

- MONK datasets(classification tasks): we compare SVM, k-NN, and Random Forest implemented within the same software tool (Scikit-learn), and Neural Network models implemented using different tools(Scikit-learn and Keras)

- ML-CUP 2025 dataset (regression tasks): we compare SVM, k-NN, Random Forest, and XGBoost implemented with Scikit-learn, and Neural Network models implemented with Scikit-learn, Keras and Pytorch and apply the selected final model to the **CUP blind test set.**

Pipeline and features for the model implementation

- **MONK Pipeline**: one-hot encoding preprocessing → model training and selection using Stratified K-Fold cross-validation (k = 3) with grid search → retraining of the best model → evaluation on the provided test set.
- **ML-CUP 2025 pipeline**: data splitting (80% TR / 20% internal TS) → PCA preprocessing → model training and selection via K-Fold cross-validation (k = 5) with grid search → retraining of the best model - model evaluation on the internal TS → select best of best models → retrain on full training set → predict the CUP blind test set.

- **SVM, k-NN, RF, XGBoost** models were trained using library implementations, with hyperparameters selected via grid search and evaluated under the same validation schema.
- **Neural Networks:** we adopted architectures with one hidden layer (2–4 units for MONK), 1–2 layers for CUP, tanh or ReLU as activations, and sigmoid output for classification, linear output for regression. Training was performed using SGD or Adam optimizer with mini-batches, random weight initialization multiple run, L2 (Tikhonov) regularization, and early stopping based on validation loss, with fixed maximum epochs as stop condition

Monk Result: Best Hyperparameters on KNN, SVM, RF

Task	KNN	SVM	Random Forest
Monk 1	n_neighbors: 23, metric: manhattan, weights: distance	C: 1, coef0: 0, gamma:1, degree: 2 kernel: poly	n_estimators: 50, max_depth: 7, max_features: None, min_samples_leaf: 1, min_samples_split: 2
Monk 2	n_neighbors: 24, metric: manhattan, weights: distance	C: 10, coef0: 1, gamma:1, degree: 2 kernel: poly	n_estimators: 50, max_depth: 12, max_features: sqrt, min_samples_leaf: 1, min_samples_split: 2
Monk 3	n_neighbors: 11, metric: manhattan, weights: distance	C: 1, coef0: 1, gamma:0.1, degree: 4 kernel: poly	n_estimators: 200, max_depth:5, max_features: None, min_samples_leaf: 1, min_samples_split: 2

Monk Results : KNN, RF, SVM Performances

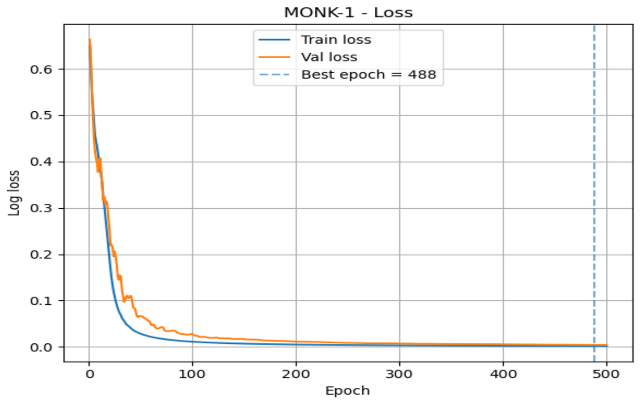
For each best model we evaluate on test set: accuracy, classification report, confusion matrix, ROC curve

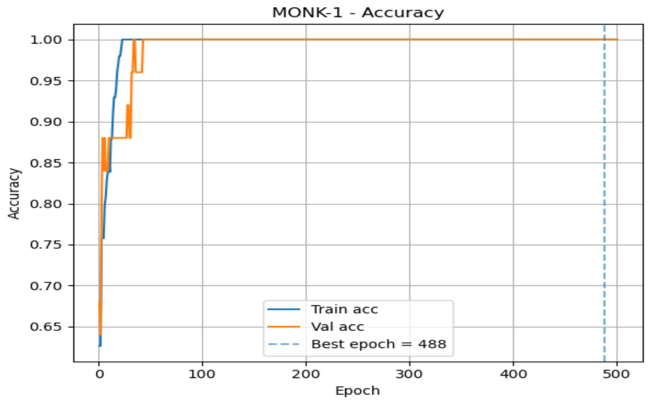
	Accuracy(%)			Macro Avg f1-score(%)			ROC AUC(%)		
Task	KNN	RF	SVM	KNN	RF	SVM	KNN	RF	SVM
Monk 1	80.79	96.30	100	80.63	96.29	100	91.3	99.7	100
Monk 2	78.94	72.92	100	73.68	69.37	100	85.3	82.7	92.8
Monk 3	90.97	97.45	98.61	90.94	97.45	98.61	95.1	99.9	99.9

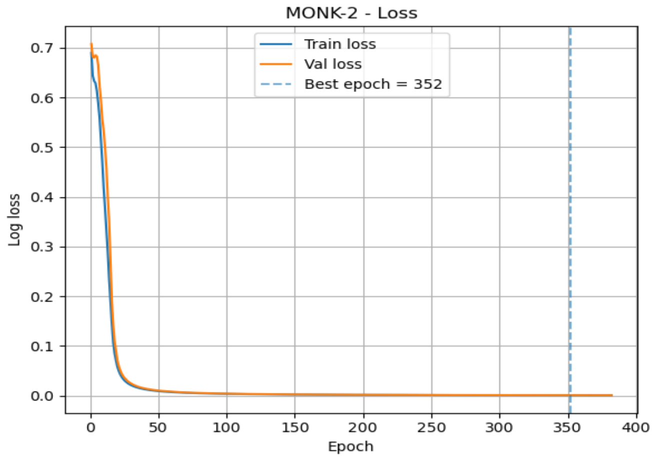
Monk Results Best Hyperparameters of Neural Network

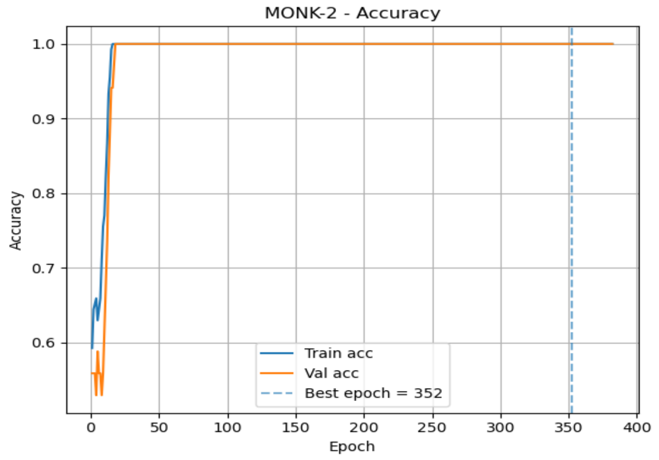
Task	Scikit Learn	Keras	Loss (TR/TS)	Accuracy (TR/TS)
Monk 1	unit=3, act=tanh, η_init =0.05, μ=0.9, batch=16	unit=3, act=tanh, η=0.1, μ=0.7, batch =16	0.0016 /0.0022 0.0032/0.0046	100%/100%
Monk 2	unit=3, act=tanh, η_init=0.05, μ=0.9, batch=8	unit=3, act=ReLU, η=0.01, μ=0.7, batch= 16	0.0007/0.0008 0.0363/0.0397	100%/100%
Monk 3 (no reg)	unit=3, act=tanh, η_init=0.05, μ=0.9, batch=8	unit=3, act=tanh, η=0.01, μ=0.5, batch= 16	0.0968/0.1099 0.1331 /0.1484	97.54%/96.53% 95.90%/95.14%
Monk 3 reg	unit=3, act=tanh, η_init=0.005, μ=0.9, λ=0.5 batch =8	unit=2, act=tanh, η=0.01, μ=0.5, λ=0.1, batch =16	0.3654/0.3414 0.340/0.3011	93.44%/97.22% 93.44%/97.22%

Learning curve of Monk 1 & 2

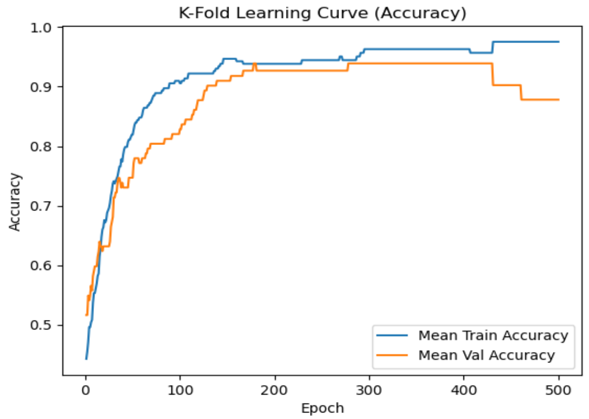


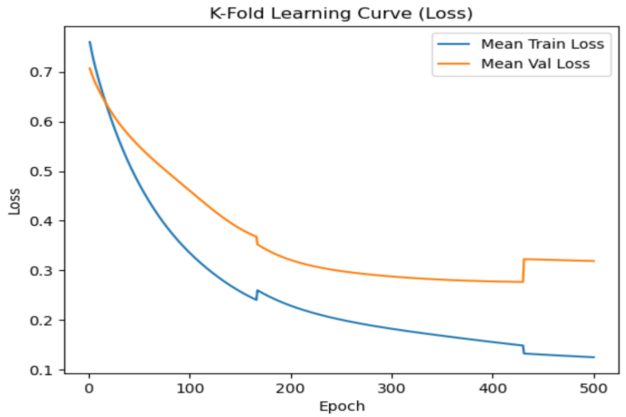


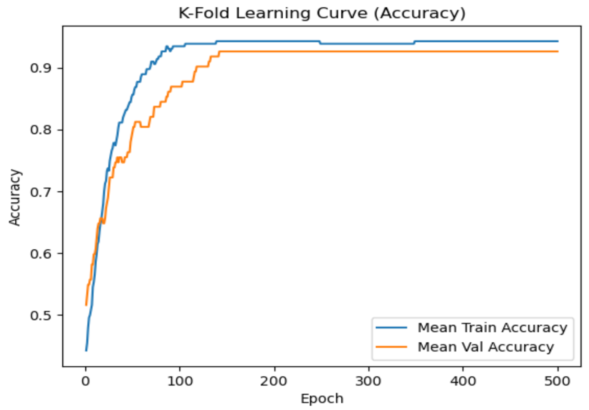


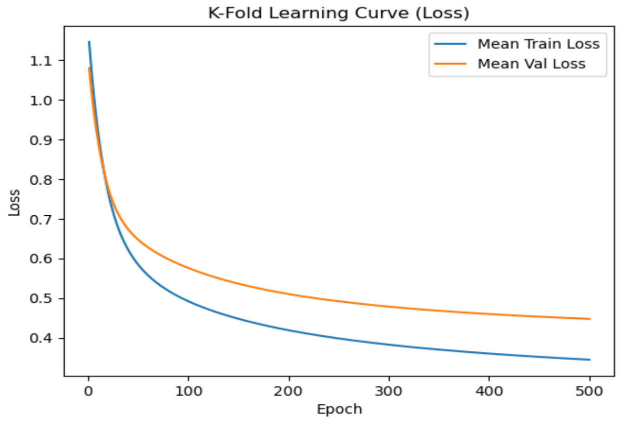


Learning curve of Monk 3 (with and without reg)



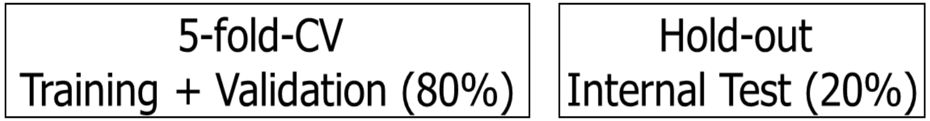






CUP Validation schema: data splitting

We split the dataset provided in TR and TS (as below). We use KFold CV for ML-Cup, with a number of folds equal to 5. We use the method from the sklearn library that allows us to shuffle the records before splitting them into folds. This approach is used for all methods from KNN, Random Forest, XGBoost, SVR, Neural Networks. Finally, we retrain the best model over the whole training set.



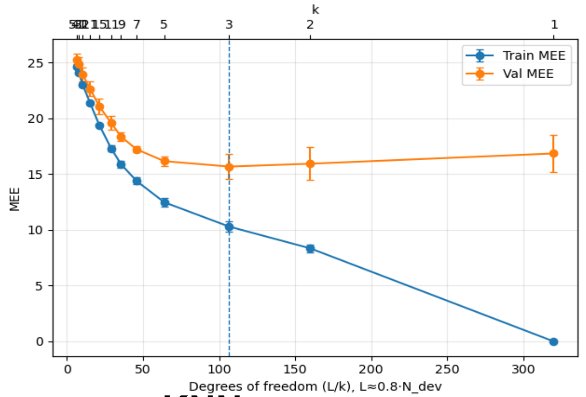
CUP Validation schema: model selection

For KNN Regressor, we use Degrees of freedom plot to choose best possible k. For SVR, Random Forest, XGBoost, we adjust each hyperparameter (such as C, gamma, epsilon for SVR) individually while keeping the others fixed. After evaluating performance through cross-validation, we select the best value for each hyperparameter. We also monitor both train MEE and test MEE to ensure there is no overfitting. These values are then used as the basis for the subsequent, more detailed grid search.

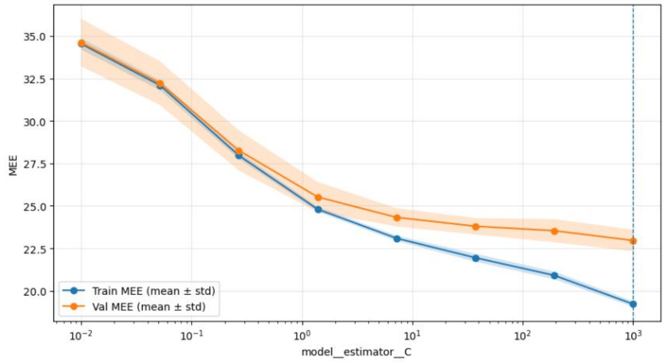
Model	Hyper parameters range
KNN	K_neighbors: (5, 7, 9, 11, 15, 21, 31, 41, 51); Knn_weights: (uniform, distance) Knn_p: (1, 2)
SVR	gamma: [0.1, 0.3, 0.37, 0.4, 0.5] C: [30, 50, 80, 120, 100, 300, 500, 1000] epsilon: [0.3, 0.6, 1.0, 1.5, 2.0, 3.0, 4.0],
Random Forest	n_estimators: [100, 200], max_depth: [3], min_samples_leaf: [1, 2, 4], max_features: [0.5, 1.0]
XGBoost	max_depth: [4, 5, 6]; n_estimators: [60, 80] min_child_weight [5, 10, 20]

CUP Validation schema: model selection

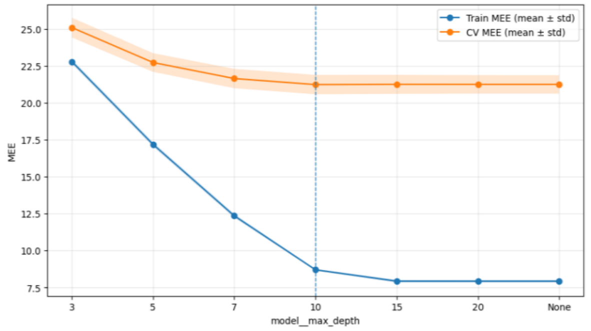
Below are some plots to see best range of hyper parameters for each algorithm (more detail at Appendix)



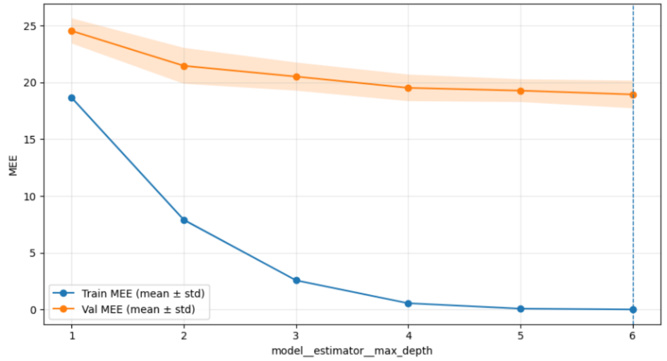
KNN regressor



SVR



Random Forest regressor



XGBoost

CUP Validation schema: model selection

For Neural Networks, we run multiple GridSearch, keeping the number of epochs fixed at 100 to 200 (for Pytorch and Keras) and 500 for Scikit learn in order to compare results in terms of loss. With an initial set of parameters for grid search, we narrow them down after each result from the grid search to reduce running time and produce results. (For example, after multiple runs, noticing that the Adam optimizer performs better, we eliminated SGD and focused on Adam).

Library	Hyper parameters range
Scikit learn	hidden_layer_sizes: [(32,), (64,32), (32, 16), (64, 64), (128, 64)] batch_size: [128, 256], learning_rate_init: [1e-3, 2e-3, 3e-3, 5e-3], alpha: [0.01], beta_1: [0.96, 0.95, 0.99, 0.9], beta_2: [0.999, 0.9999], epsilon: [1e-8, 1e-7], activation: ["relu", "tanh"]
Py torch	hidden_layer_sizes: [(32,), (64,32), (32, 16), (64, 64), (128, 64)] batch_size: [128, 256], learning_rate: [1e-3, 2e-3, 3e-3, 5e-3], beta_1: [0.96, 0.95, 0.99, 0.9], beta_2: [0.9995, 0.9999], epsilon: [1e-8, 1e-7], activation: ["relu", "tanh"], weight_decay: [0.01, 1e-5, 1e-4, 1e-3]
Keras	hidden_layer_sizes: [(32,), (64,32), (32, 16), (64, 64), (128, 64)] batch_size: [32, 64], learning_rate: [0.01, 0.1], activation: ["relu", "tanh"], weight_decay: [0.01, 0.1]

CUP Results (KNN, SVR, XGBoost, Random Forest)

We identified the best-performing hyperparameter configuration for each model by gridsearch. To mitigate overfitting, hyperparameter selection was constrained to previously validated “good” regions of the search space and finalized based on cross-validated performance rather than training metrics alone. Both KNN and SVR exhibited relatively short runtimes, primarily because KNN was optimized with a small number of neighbors, and SVR was fixed RBF kernel. For the ensemble-based models, namely XGBoost and Random Forest (RF), runtime scaled with the number of estimators (i.e., the number of trees/boosting rounds).

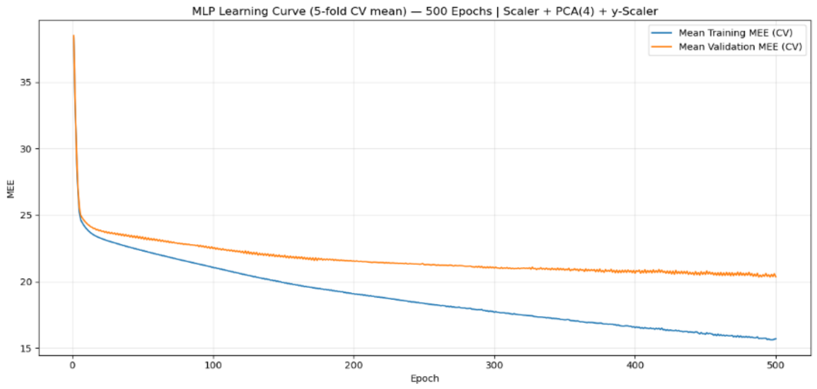
Model	Best hyper parameters	MEE_TR	MEE_VAL	MEE_TEST
KNN	N_neighbors: 5; p: 2; weights: distance	0	15.44	14.38
SVR	C: 500, epsilon: 2.0, gamma: 0.5	13.15	18.22	18.13
XGboost	max_depth: 2, min_child_weight: 20, n_estimators: 30	17.95	22.81	22.04
RF	max_depth: 5, max_features: 1.0, min_samples_leaf: 1, n_estimators: 200	17.97	22.88	22.53

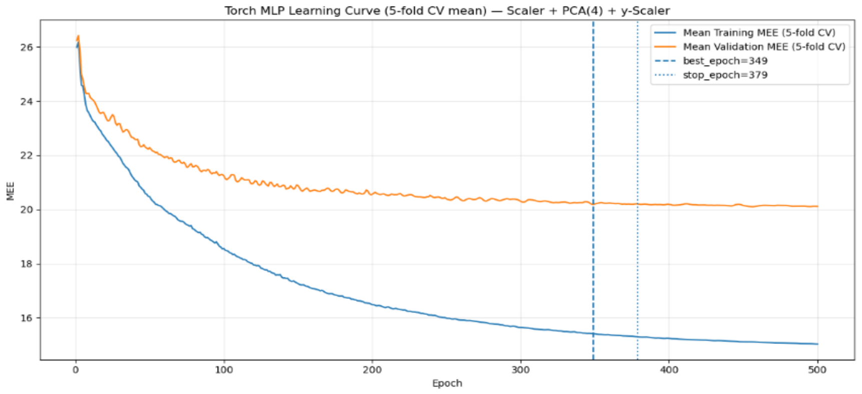
CUP Results (Neural Netwworks)

We evaluated SGD and Adam while training epochs are 100 for pytorchand 500 for sklearn and keras. In parallel, we compared different batch sizes and found that batch size as 128 and 32 yielded the best so subsequent experiments fixed the batch size at 128 and 32 to speed up experimentation.
After fixing the batch size, Adam consistently outperformed SGD, so the detailed evaluation phase focused on Adam. During architecture testing, the best-performing multilayer perceptron (MLP) configurations were: [32], [64, 32], [64, 64], and [128, 64]. We then narrowed the search space and concentrated the grid around these candidate architectures for fine-tuning.
For learning-curve, we fixed epochs as 500; and applied L2 regularization.

Library	Best Hyperparameters	MEE_TR	MEE_VAL	MEE_TEST
Scikit learn	hidden_layer_sizes: (128, 64), activation: 'relu', batch_size: 128, optimizer: 'adam’, lambda: 0.0001, eta_init: 0.005, beta1: 0.95, beta2: 0.9999	15.69	20.37	20.71
Pytorch	hidden_layer_sizes: (128, 64), activation: 'relu', batch_size: 128, optimizer: 'adam’, lambda: 0.0001, eta: 0.005, beta1: 0.95, beta2: 0.9999	15.41	20.19	18.57
Keras	hidden_layer_sizes: (64, 64), activation: 'relu', batch_size: 32, optimizer: 'adam’, lambda: 0.01, eta: 0.01,	12.98	18.48	17.26

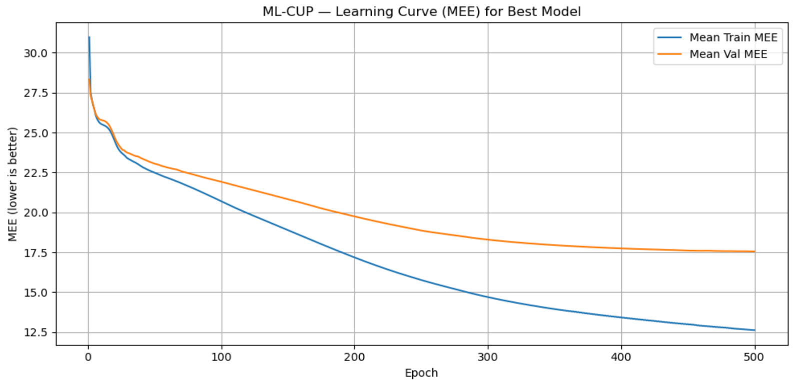
learning curve Neural networks





Scikit learn - Adam

Pytorch - Adam



Keras - Adam

CUP Results (best model)

- Finally, we selected KNN Regressor as the best-performing model for predicting the CUP target, as it achieved the lowest internal test MEE and also provided substantially better runtime efficiency compared with all other evaluated models. Potential overfitting was assessed and mitigated using the degree-of-freedom plot, which guided the choice of model complexity and supported a better generalization trade-off.
- The KNN model achieves train MEE = 0 because predictions are computed on the same training set and the best configuration uses weights="distance". For each training sample, KNN includes the sample itself among its nearest neighbors with distance = 0, which receives a dominating weight; therefore each point will predict itselt.

Model	**Hyper parameters range**		
KNN	**k**: (5, 7, 9, 11, 15, 21, 31, 41, 51); **Knn_weights**: (uniform, distance); **Knn_p**: (1, 2)		
Result	MEE_TR = 0	MEE_VAL = 15.44	MEE_TEST = 14.38

Discussion

- We applied an advanced method XGBoost but did not work well for CUP because ML-CUP25 has few samples and highly correlated inputs, so boosted trees are prone to high variance/overfitting and unstable split choices.
- β1 (momentum) controls how strongly Adam smooths the gradient direction: higher β1 produces smoother train/validation MEE curves.

- β2 (variance smoothing) controls how quickly Adam adapts per-parameter step sizes: higher β2 yields more stable learning curves but can slow convergence.

- Learning-curve smoothness increases with larger batch sizes because gradients have lower variance.

- PyTorch and Keras also expose learning rate directly (easier to implement modern training tricks); scikit-learn's advantage is convenience for CV/tuning.

Conclusions

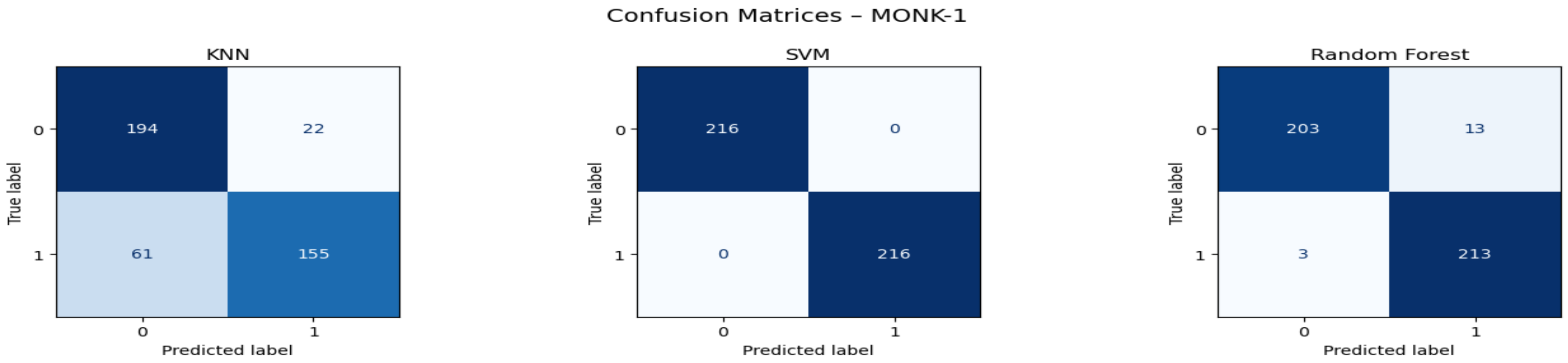
This project allowed us to gain a deeper understanding of the role of model selection, validation strategies, and hyperparameter tuning in Machine Learning. Through an extensive experimental comparison of classical models and neural networks, we observed that model complexity must be aligned with data characteristics, as advanced techniques do not necessarily guarantee better performance. Comparing different tools also highlighted trade-offs between training flexibility and experimental efficiency, strengthening our awareness of practical modeling decisions.

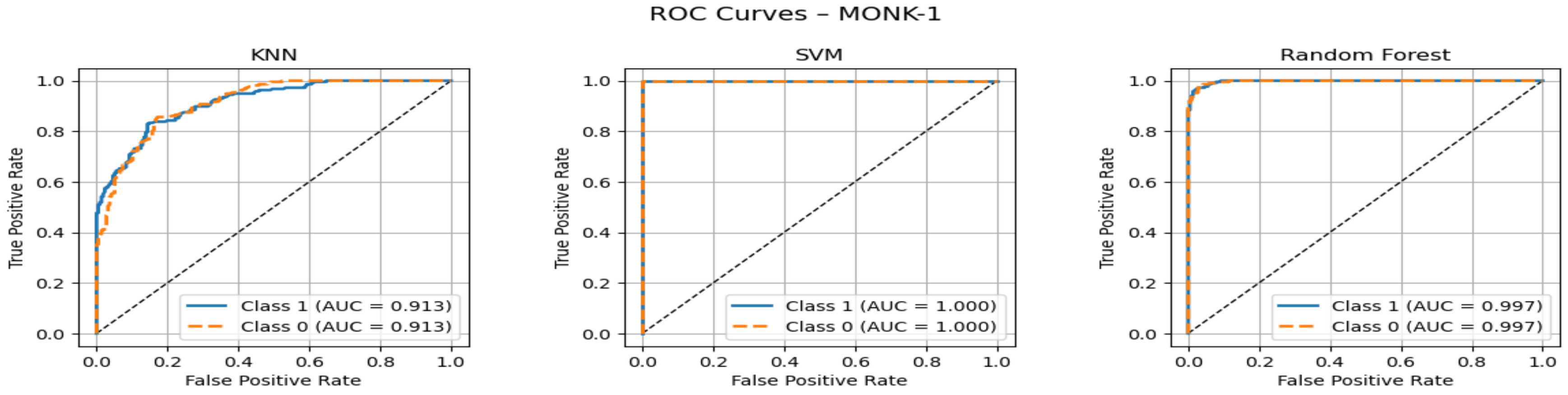
Blind Test Results:

name of the result files:Vietnamese-ML-CUP25-TS

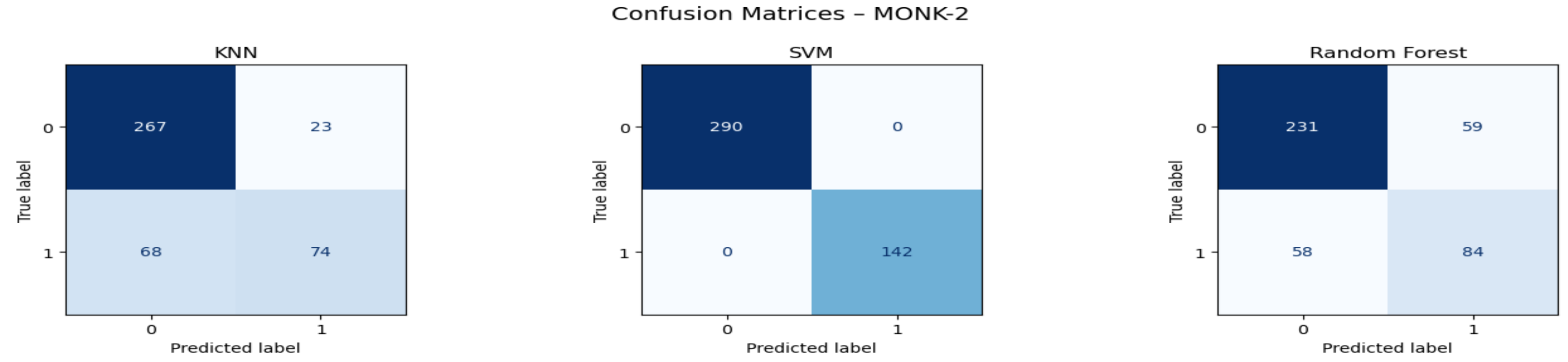
your nickname: Vietnamese

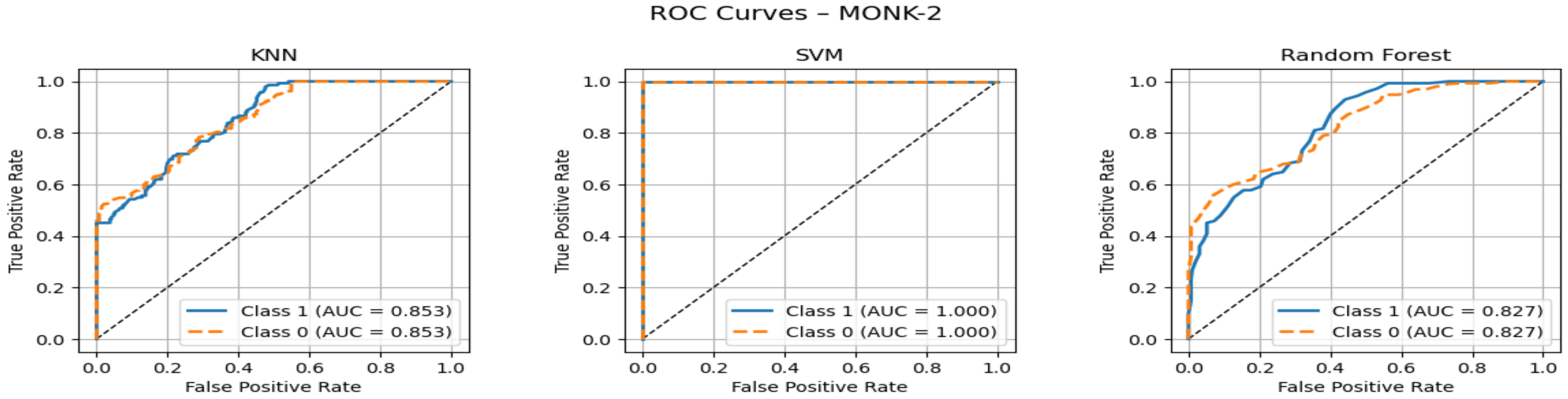
Appendix 1: Monk 1 evaluation metrics



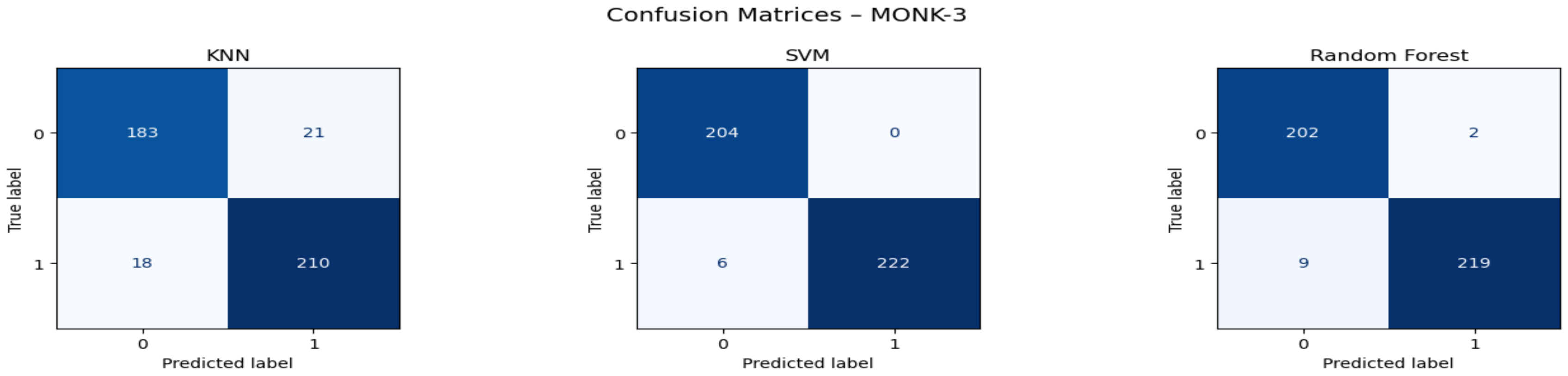


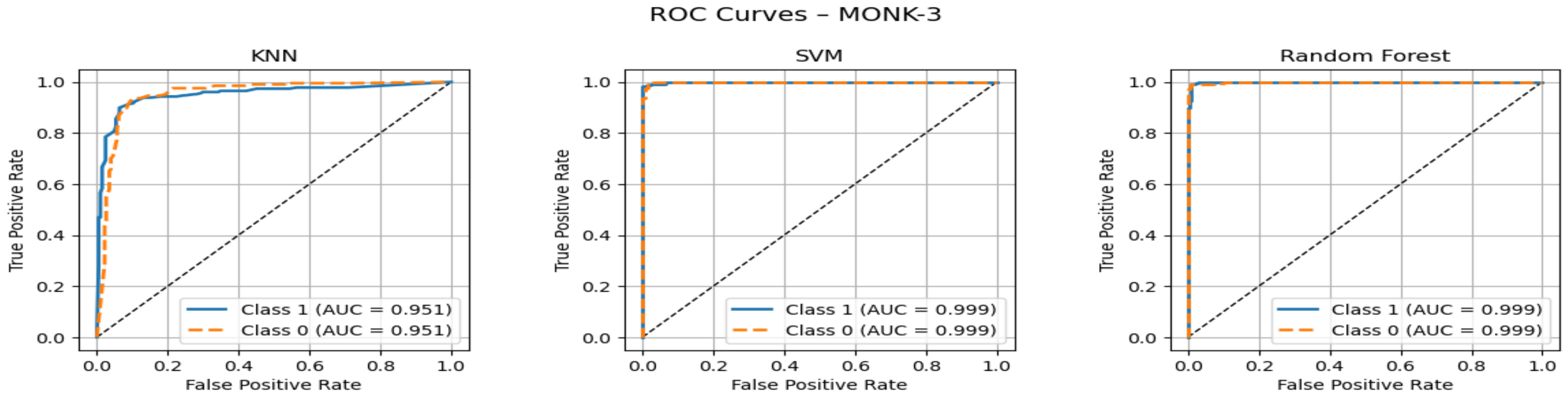
Appendix 2: Monk 2 evaluation metrics





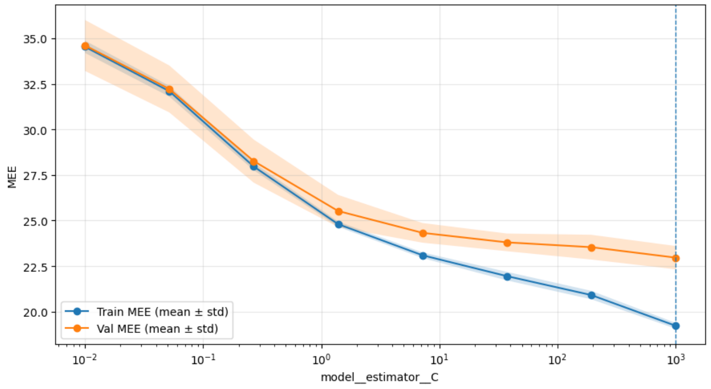
Appendix 3: Monk 3 evaluation metrics

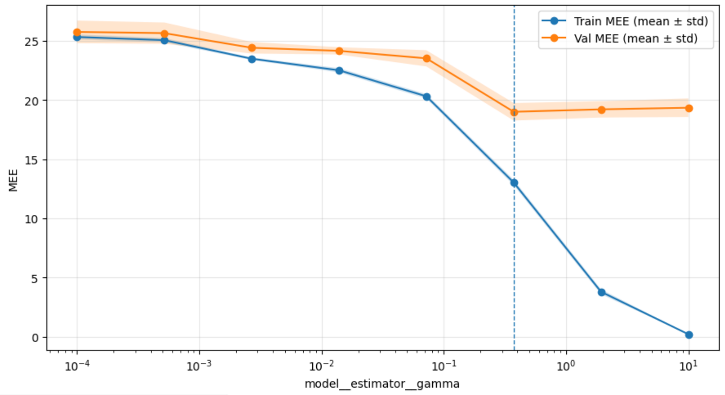


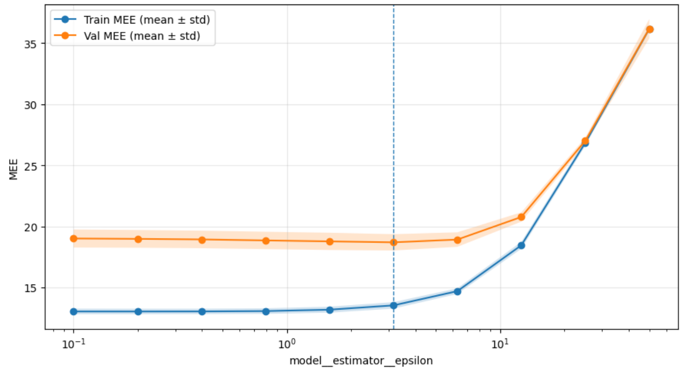


Appendix 4: SVR

Below are some plots to see best range of hyper parameters for SVR

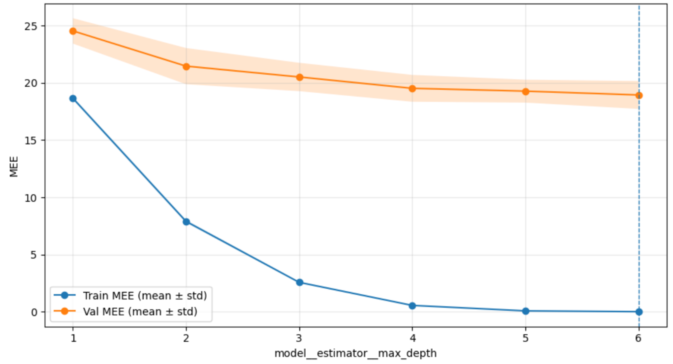


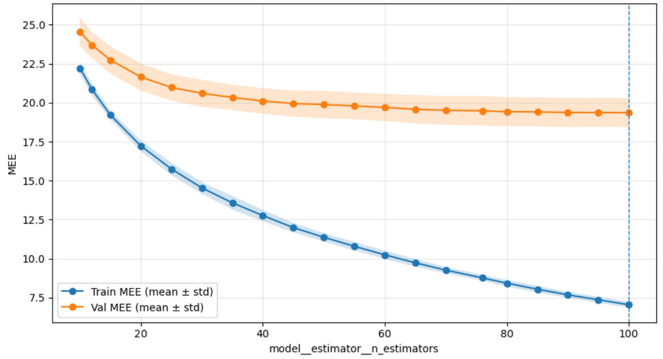


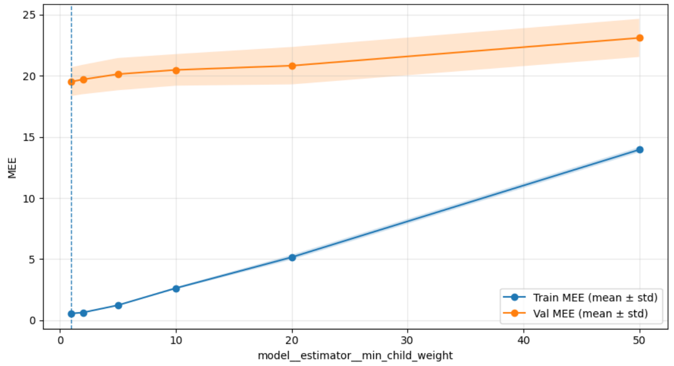


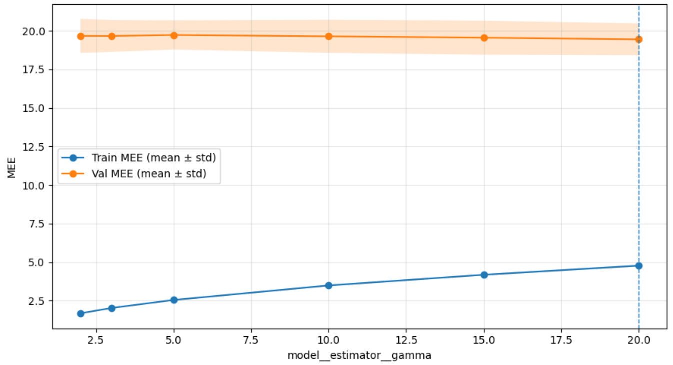
Appendix 5: XGBoost

Below are some plots to see best range of hyper parameters for XGBoost



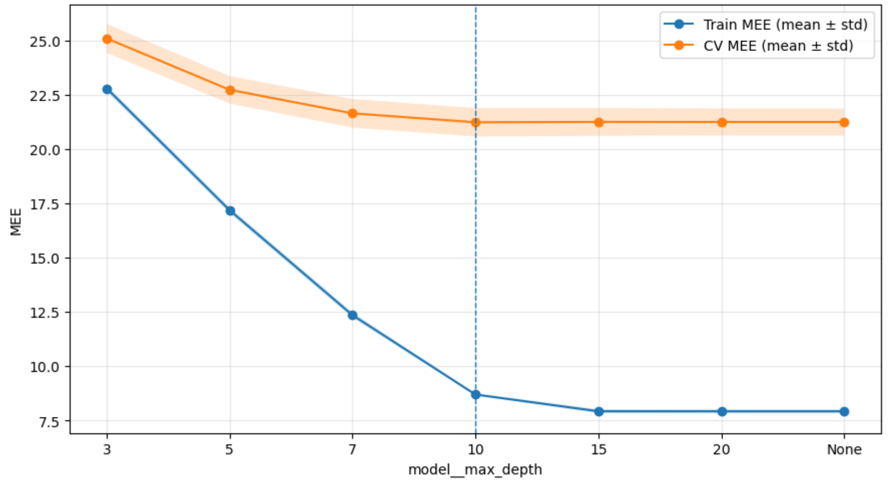


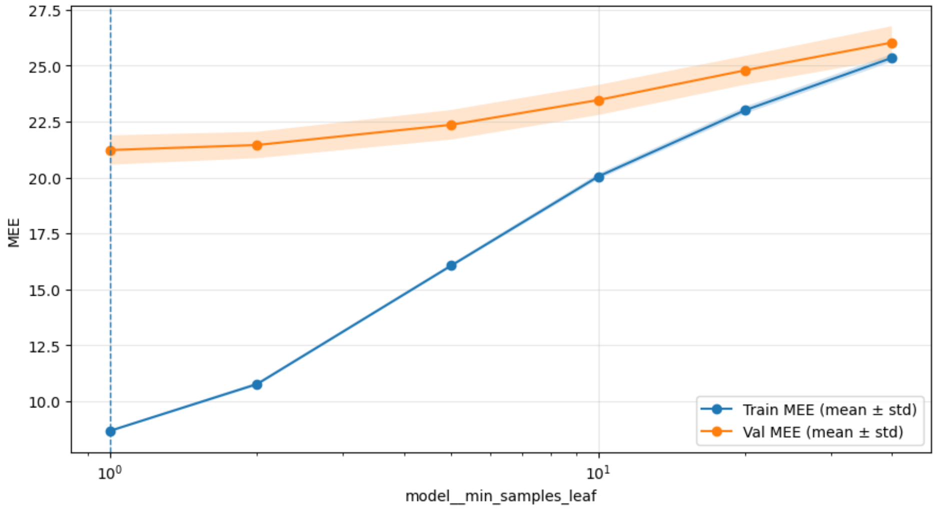




Appendix 6: Random Forest

Below are some plots to see best range of hyper parameters for Random Forest


Train MEE (mean ± std)
CV MEE (mean ± std)
MEE
model__max_depth
None


Train MEE (mean ± std)
Val MEE (mean ± std)
MEE
model__min_samples_leaf
10^0
10^1