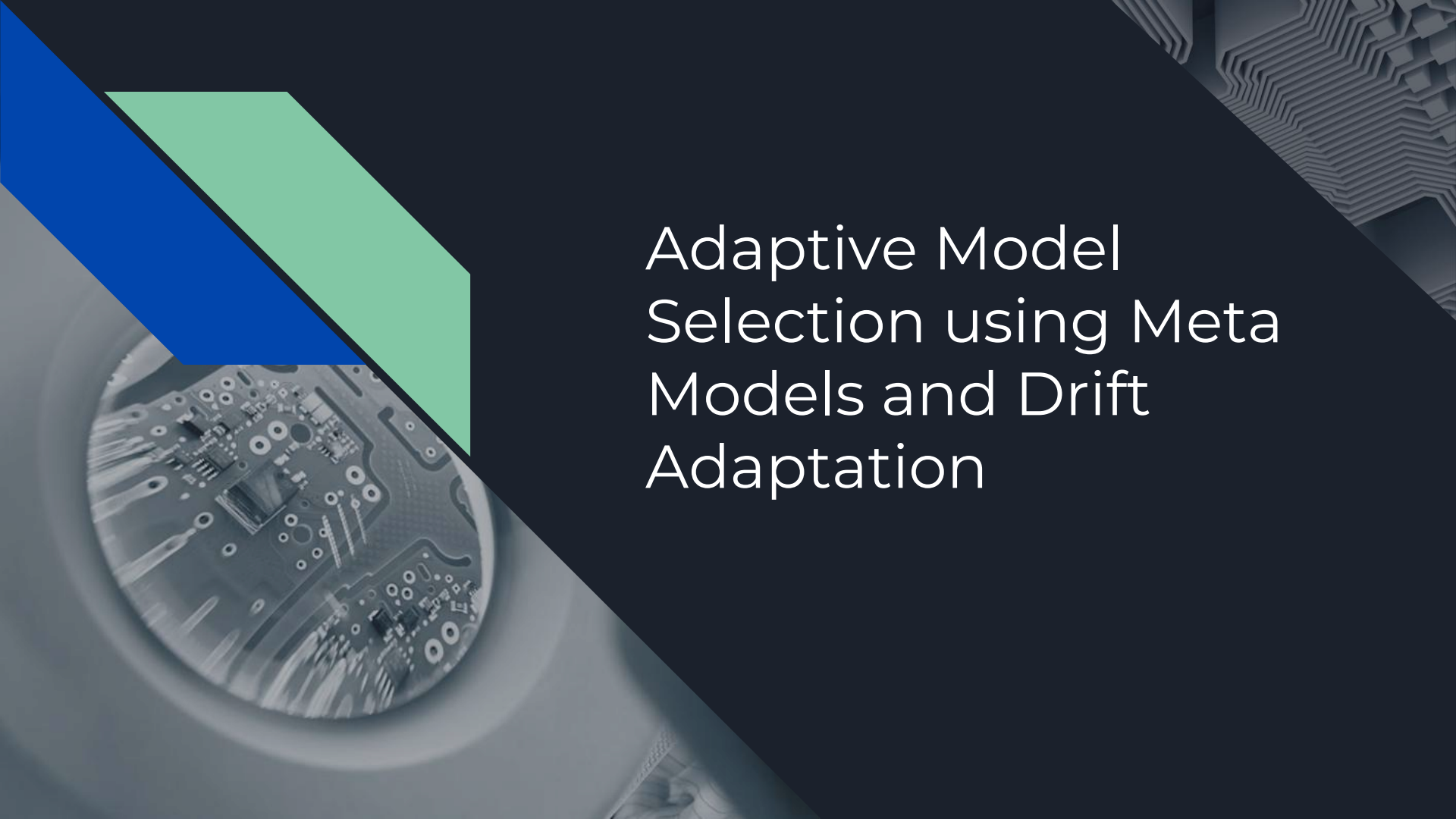
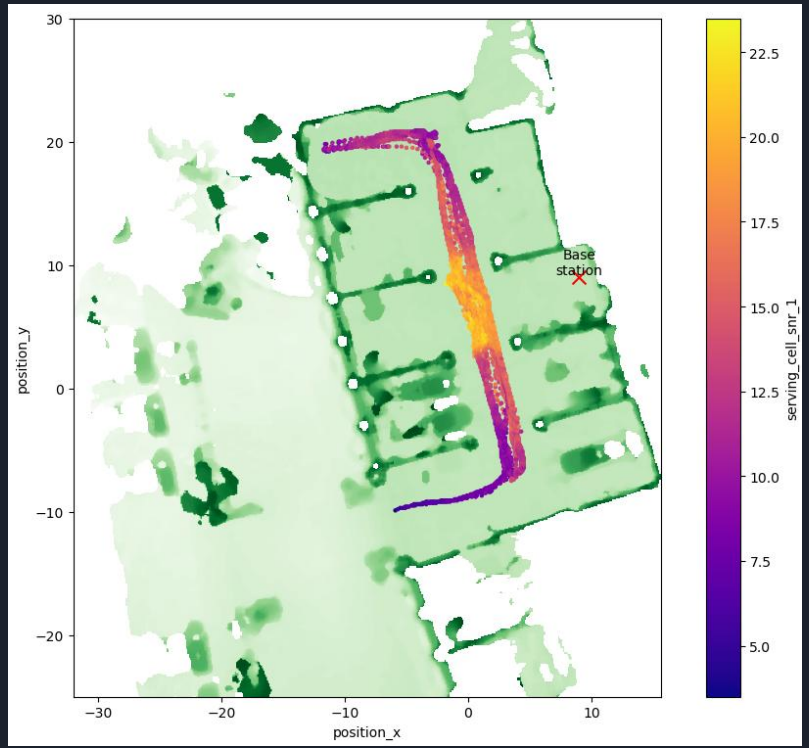


Adaptive Model Selection using Meta Models and Drift Adaptation

Problem Statement

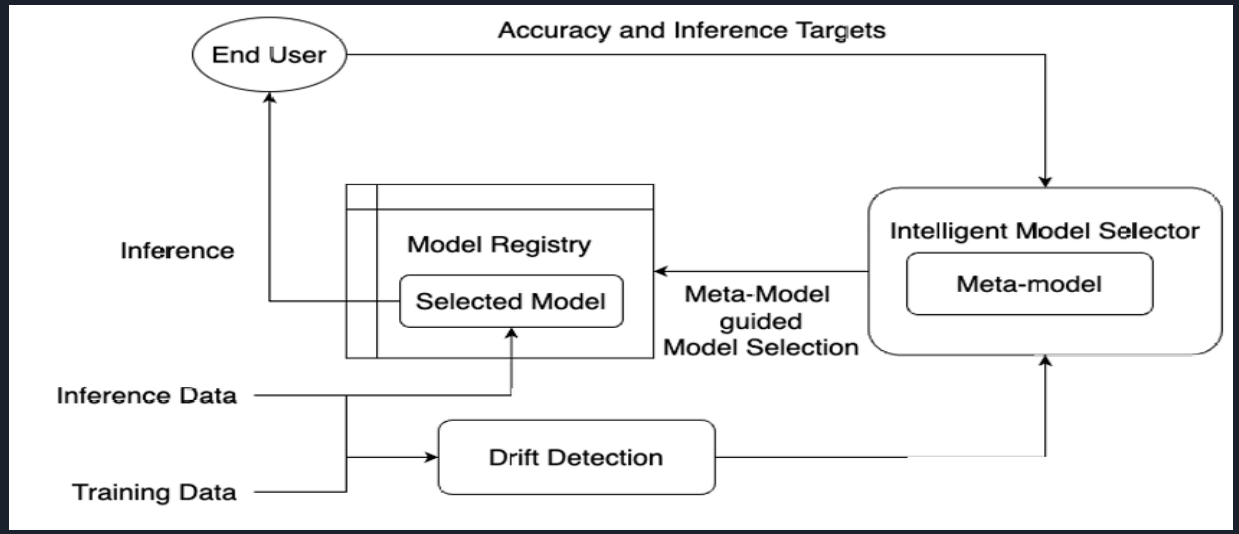
Creating an intelligent model selector based on the user query and requested targets.

The models that are used are trained on iV2I+ dataset . They are to predict the distance of AGV from the base station



Proposed Architecture

- Multi-Registry Storage: Models are pre-trained on different drift variants (e.g., X^k) and stored in specialized registries.
- Drift Detection: Incoming inference data is analyzed using MMD to route the request to the correct registry.
- Meta-Model Selection: A classifier predicts the optimal model index (i*) based on user constraints (Acc_min, T_max) without running full inference.



Drift Simulation & Registry Generation

Drift Vector (δ): We generate a fixed random direction vector sampled from a standard normal distribution to define the direction of the drift :

$$\boldsymbol{\delta} \sim \mathcal{N}(0,1)^d$$

Transformation Logic: For every sample x_i in the dataset, we apply an additive shift scaled by a magnitude scalar m : (Where m represents the severity of the shift).

$$\mathbf{x}_i' = \mathbf{x}_i + (m \cdot \boldsymbol{\delta})$$

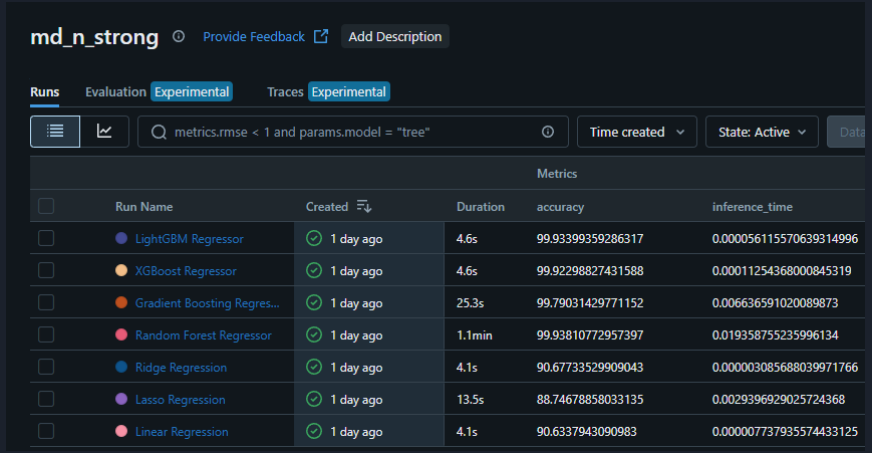
Training Registries (Known Drifts): Models were trained and stored for drift magnitudes:
m ∈ {-7, -2, 2, 7} .

Adding multiple registries

Created multiple variations of dataset. Each variation has a unique drift magnitude from the original

Aim is to select the registry which has been trained on variant of dataset which is closest to user data(inference data)

Now the meta data will also have information about the drift magnitude of model registries

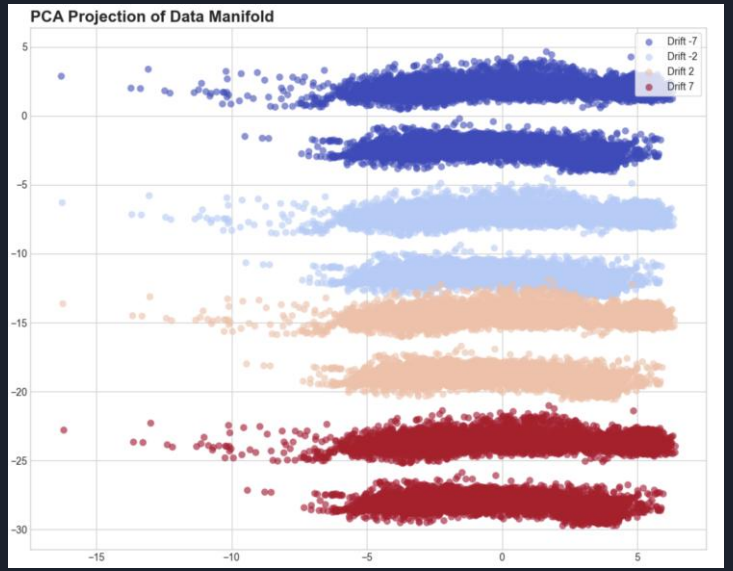


Drift Detection & Quantification

Systematic Feature Drift: We simulate Covariate Shift by applying transformations to the feature space, creating distinct statistical environments

We use MMD to measure the distance between the Inference Data and our Model Registries. This allows us to select the registry (k*) that is statistically closest to the user's input

$$k^* = \arg\min_k \ \mathrm{MMD}^2(X_{\text{user}}, X^{(k)})$$



Our drift simulation challenges static models, necessitating the adaptive registry selection mechanism.

Meta-Model Guided Selection Strategy

As the iv2I+ dataset has about 92 features and almost 53K rows, training models in the registry and the meta model on all the features may lead to performance issues.

Used Pearson correlation coefficient to understand the relationship between features and the target variable(“distance_to_bs”)

Considered the top 10 features to train the meta model on. Then, a score is given to every model in the registry based on the accuracy and inference on the given query.

The formula for Pearson's r is:

$$r = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y}$$

Where:

- $\mathrm{Cov}(X,Y)$: Covariance between variables X and Y.
- σ_X: Standard deviation of X.
- σ_Y: Standard deviation of Y.

$$S_{mm} = \frac{1}{\sum_{i=0}^{k-1} \alpha_i M_i \times \sum_{i=0}^{k-1} \beta_i O_i},$$

M is model metrics and O is operation metrics

Hardware Constraints

We evaluated inference performance on a high-end Laptop (Intel Core i7-12650H processor and 16GB RAM and an NVIDIA RTX 3050 Ti.) versus a resource constrained a Raspberry Pi 4 Model B Rev 1.5 with 4GB RAM and a 1.5GHz quad-core ARM Cortex A72 processor.

On average, the Raspberry Pi is ~6.87x slower than the laptop across all models

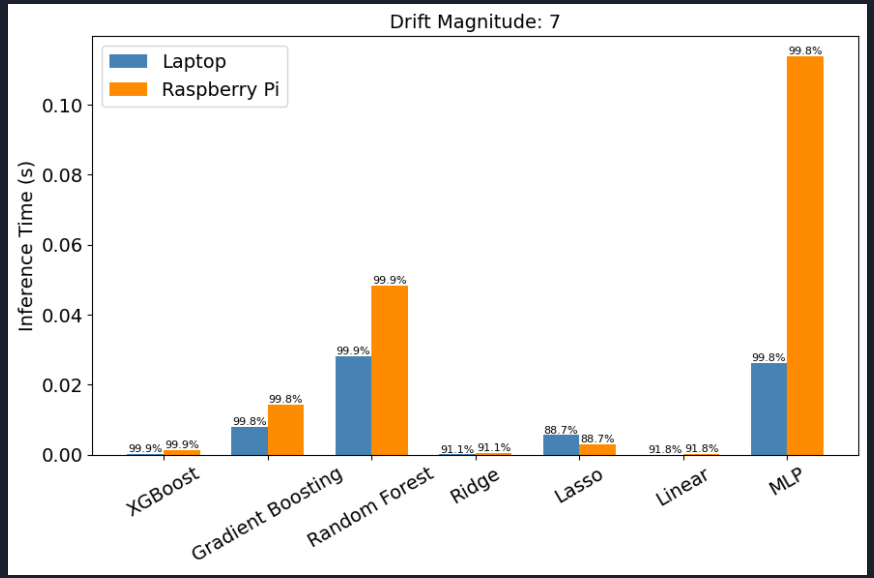
Model	Avg. Laptop Time (s)	Avg. Pi Time (s)	Pi/Laptop Ratio
XGBoost Regressor	0.00022185	0.00153750	6.93x
Gradient Boosting Regressor	0.00646750	0.01431250	2.21x
Random Forest Regressor	0.01923750	0.04871000	2.53x
Ridge Regression	0.00002433	0.00030877	12.69x
Lasso Regression	0.00418750	0.00551000	1.32x
Linear Regression	0.00001623	0.00025875	15.95x
Deep Scikit MLP Regressor	0.01675000	0.10781000	6.44x
Overall Average	**0.00670070**	**0.02549250**	**6.87x**

Experimental Results: Accuracy vs. Latency

Robustness: Accuracy remains stable across drift magnitudes.

Latency Spike: Complex models (e.g., MLP, Random Forest) see massive latency spikes on the Raspberry Pi.

Generalization to Unseen Data: When tested on a drift magnitude of 4 (which was not in our training set), the MMD metric correctly identified the "Drift 2" registry as the closest statistical match (MMD=7.43 vs 225.97 for Drift 7)

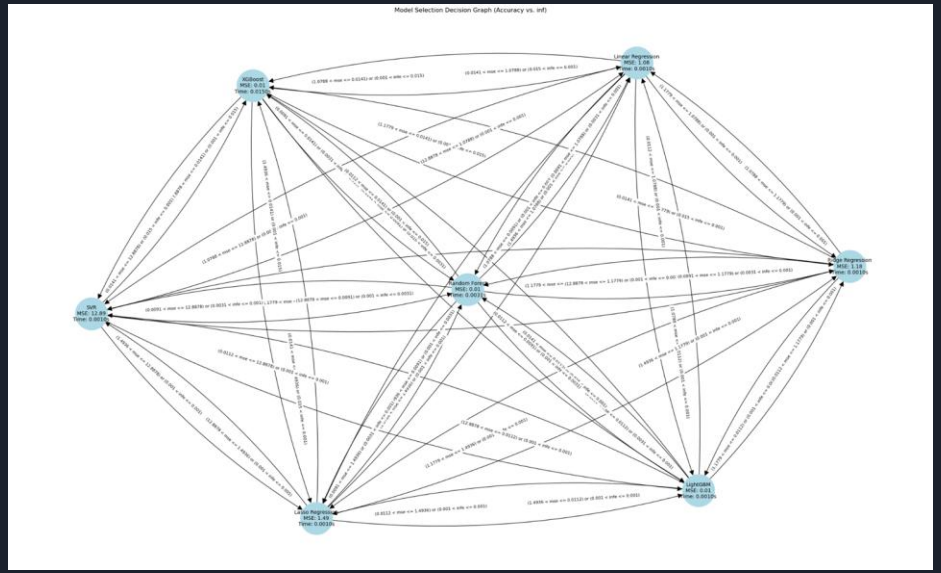


Graph approach

A Directed Acyclic Graph(DAG) with Models as nodes and Accuracy and inference conditions as edges has been created.

Used a greedy approach where the model searches for its neighbourhood nodes and checks whether the current node has more inference and Mean square error.

Didn’t directly put hard values onto the edges but created a bound to traverse the graph.



The model graph

Graph approach(contd.)

```
42  # Example usage
43  new_X = X_test[0:1]  # Simulate new input
44  best_model, path = select_model_graph(new_X, G)
45  print(f"Selected Model: {best_model}")
46  print(f"Decision Path: {path}")
47
```

[97] ✓ 0.0s

```
Selected Model: Random Forest
Decision Path: ['Linear Regression', 'Ridge Regression', 'Random Forest']
```

Run Name	Created	Duration	accuracy	inference_time
LightGBM Regressor	4 minutes ago	10.1s	0.9991668202852793	0.000005193273036451702
XGBoost Regressor	4 minutes ago	8.4s	0.9989567869696674	0.0000064368060660496045
Random Forest Regressor	4 minutes ago	44.4s	0.9993266321778863	0.000061884711373994 62
Support Vector Regression	5 minutes ago	15.7s	0.043502245542546336	0.000647128730968982
Lasso Regression	5 minutes ago	8.0s	0.8891526376630461	4.1890506114020117e-7
Ridge Regression	5 minutes ago	5.1s	0.9125813746554263	8.225961327181118e-7
Linear Regression	5 minutes ago	6.2s	0.9199320598994093	0.000006992884818096712

Decision Graph

Testing with KL divergence

- KL divergence measures how much one probability distribution differs from another. A higher KL divergence means more drift.

```
# Define drift levels
drift_levels = {
    "org": ("multi_drift_org", 0.0),
    "mild": ("multi_drift_mild", 0.3),
    "severe": ("multi_drift_severe", 0.5)
}
```

- Introduced a metric which measures the drift score for each registry

```
for col in df_original.select_dtypes(include=[np.number]).columns:
    if col!=target_column:
        noise = np.random.normal(0, drift_magnitude * df_original[col].std(), len(df_original))
        df2_user[col] += noise

print(f"Random drift applied with magnitude: {drift_magnitude}")
```

✓ 0.0s

```
Random drift applied with magnitude: 0.24870121417508897
```

- To select the best model from the selected registry, we use the meta model approach.

```
else:
    selected_registry = "severe"

print(f"User data best matches with {selected_registry} registry.")
```

✓ 1.9s

```
C:\Users\mvska\AppData\Local\Temp\ipykernel_15016\3054528118.py:12: Runtim
  kl_div = entropy(np.histogram(original_df[col], bins=20, density=True)[0
25424.28119744915 16949.434527408463 42373.53386413355
User data best matches with mild registry.
```

Future Work

- Including more datasets with different distributions
- Adding better usage of bias towards important features (Like using pearson correlation coefficient)
- Adding more complex models into the registry so that inference also becomes a important metric
- Adding selective retraining if the drift in user data varies too much from the registries

References

- Model selection via Meta learning
- Which model is best for my data
- When and how to update AI/ML models in 6G resource-constrained network domains?

Meta model approach(contd.)

To increase the overall accuracy to predict the best model, two meta models have been created. One for accuracy and the other for inference.

Each model gives an individual score for accuracy and inference. With this approach, accuracy of the meta models(acc and inf) have been boosted to 93% and 97% respectively.

```python
# Example 1: Prioritize accuracy
best_model_name = select_best_model(new_X, target="accuracy")
print(f"Best model for accuracy: {best_model_name}")
# print(f"Prediction: {best_model.predict(new_X)}")

# Example 2: Prioritize speed
best_model_name = select_best_model(new_X, target="speed")
print(f"Best model for speed: {best_model_name}")
# print(f"Prediction: {best_model.predict(new_X)}")
```

✓ 0.0s

```
Best model for accuracy: XGBoost Regressor
Best model for speed: Lasso Regression
```

```python
# Train meta-model
meta_features = meta_X_acc.drop(columns=['best_model'])
meta_labels = meta_X_acc['best_model']

meta_X_acc_train, meta_X_acc_val, meta_y_acc_train, meta_y_acc_val = train_test_split(
    meta_features, meta_labels, test_size=0.3, random_state=42
)

meta_model_1 = GradientBoostingClassifier()
meta_model_1.fit(meta_X_acc_train, meta_y_acc_train)

# Evaluate meta-model
meta_accuracy_acc = meta_model_1.score(meta_X_acc_val, meta_y_acc_val)
print(f"Meta-model acc accuracy: {meta_accuracy_acc:.2f}")
```

[44] ✓ 8.7s

```
Meta-model acc accuracy: 0.93
```

```python
# Train meta-model
meta_features = meta_X_inf.drop(columns=['best_model'])
meta_labels = meta_X_inf['best_model']

meta_X_inf_train, meta_X_inf_val, meta_y_inf_train, meta_y_inf_val = train_test_s
    meta_features, meta_labels, test_size=0.3, random_state=42
)

meta_model = GradientBoostingClassifier()
meta_model.fit(meta_X_inf_train, meta_y_inf_train)

# Evaluate meta-model
meta_accuracy = meta_model.score(meta_X_inf_val, meta_y_inf_val)
print(f"Meta-model inf accuracy: {meta_accuracy:.2f}")
```

[37] ✓ 2.8s Python

```
Meta-model inf accuracy: 0.97
```