INNOVATION (Phase 2)

DEFINITION OF HYPERPARAMETER :

Hyperparameter tuning allows data scientists to tweak model performance for optimal results. This process is an essential part of machine learning, and choosing appropriate hyperparameter values is crucial for success. For example, assume you're using the learning rate of the model as a hyperparameter.

ABSTRACT :

In the pursuit of enhancing prediction model performance, advanced techniques like hyperparameter tuning and feature engineering stand out as indispensable tools. Hyperparameter tuning allows for the optimization of parameters that aren't learned during training, such as learning rates and regularization strengths, employing methods like grid search or random search to unearth the optimal combination. Meanwhile, feature engineering involves the creation or transformation of features to more accurately represent underlying data patterns, employing techniques like scaling and dimensionality reduction. The synergy of these techniques, when iteratively explored, can lead to substantial improvements in a prediction model's accuracy and generalization capabilities.

HYPERPARAMETER TUNING ADVANCE TECHNIQUES :

There are several advanced techniques for hyperparameter tuning, some of which go beyond the basics like grid search or random search. Here's an overview of advanced techniques, each explained in a few lines:

1. **Bayesian Optimization**: Bayesian optimization models the objective function and uses probabilistic models to decide where to explore next. It's more efficient than grid or random search.

2. **Gradient-Based Optimization**: Some hyperparameters can be tuned using gradient-based optimization methods like gradient descent, particularly in deep learning.

3. Genetic Algorithms: Genetic algorithms mimic natural selection and evolve a population of hyperparameter configurations over several generations to find the best combination.

4. **Particle Swarm Optimization (PSO)**: PSO optimizes a population of potential solutions by having them "swarm" together towards the best configuration.

5. **Simulated Annealing**: Inspired by the annealing process in metallurgy, it starts with a high "temperature" (allowing exploration) and gradually decreases it to find the best configuration.

6. **Bayesian Optimization with Gaussian Processes (BO-GP)**: Combines Bayesian optimization with Gaussian processes to model the objective function and make informed decisions on where to explore.

7. **Neural Architecture Search (NAS)**: NAS automates the process of finding the optimal neural network architecture and hyperparameters, often using reinforcement learning or evolutionary algorithms.

8. **Ensemble-Based Tuning**: Optimize ensembles of models with different hyperparameter settings to leverage their combined performance.

9. **Meta-Learning for Hyperparameter Tuning**: Train a meta-model to predict the best hyperparameters based on past experiments, effectively learning from previous tuning attempts.

10. **Sequential Model-Based Optimization (SMBO)**: SMBO builds a surrogate model of the objective function and chooses the next configuration based on its predictions.

11. **Bayesian Neural Networks**: Extend Bayesian optimization techniques to tune the hyperparameters of neural networks themselves, providing more robust optimization.

12. **Random Forests for Hyperparameter Tuning**: Employ random forests to predict which hyperparameters are most influential and focus the search on those.

13. **Early Stopping**: Although not a hyperparameter tuning technique, it's essential to prevent overfitting and save time during training.

14. **AutoML Platforms:** Utilize automated machine learning platforms like Google AutoML, H2O.ai, or Auto-Sklearn that handle much of the hyperparameter tuning process.

15. **Probabilistic Programming:** Use probabilistic programming libraries like PyMC3 or Edward to model hyperparameter tuning as a probabilistic inference problem.

These advanced techniques often require a good understanding of the underlying optimization principles and may require more computational resources, but they can significantly improve the performance of your machine learning models.

STEP TO PERFORM HYPERPARAMETER TUNING :

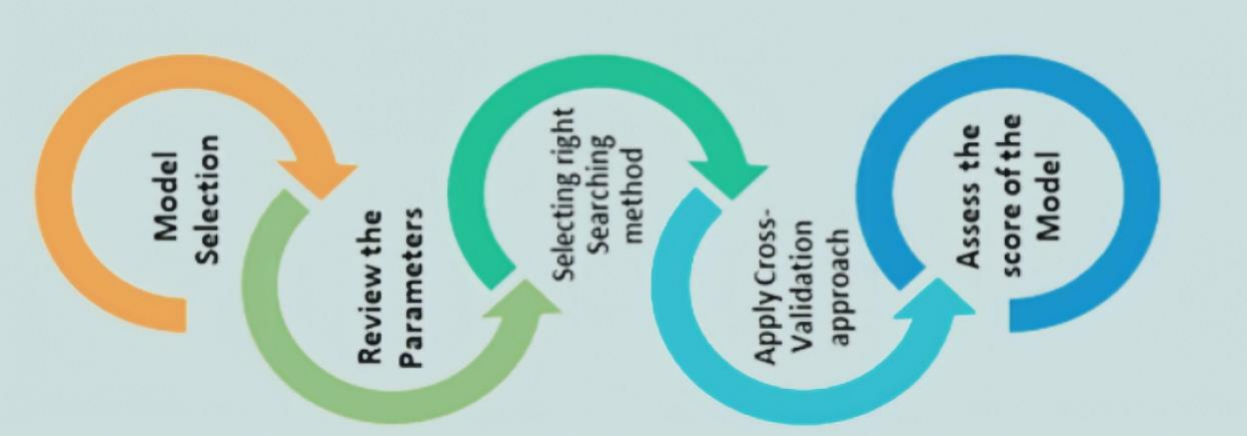
Select the right type of model.

Review the list of parameters of the model and build the HP space.

Finding the methods for searching the hyperparameter space.

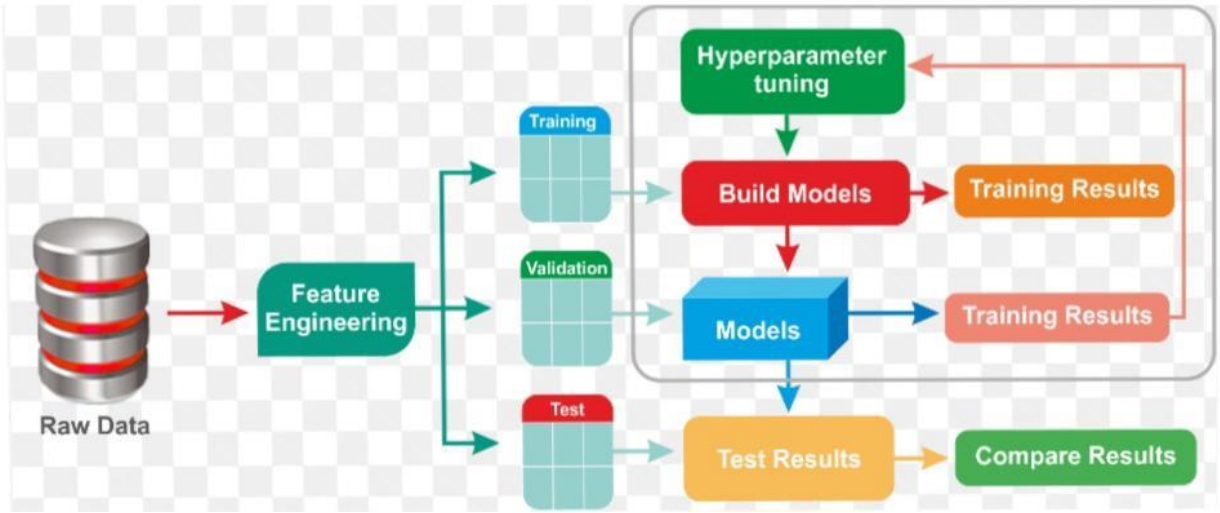
Applying the cross-validation scheme approach.

Assess the model score to evaluate the model.



TRAIN, TEST SPLIT ESTIMATOR :

With the help of this, we use to set the test and train size for the given dataset and along with random state, this is permutations to generate the same set of splits., otherwise you will get a different set of test and train sets, tracing your model during evaluation is bit complex or if we omitted this system will generate this number and leads to unpredictable behaviour of the model. The random state provides the seed, for the random number generator, in order to stabilize the model.



CODING : 1

```
# Necessary imports

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import GridSearchCV

# Creating the hyperparameter grid

c_space = np.logspace(-5, 8, 15)
```

```
param_grid = {'C': c_space}

# Instantiating logistic regression classifier

logreg = LogisticRegression()

# Instantiating the GridSearchCV object

logreg_cv = GridSearchCV(logreg, param_grid, cv = 5)

logreg_cv.fit(X, y)

# Print the tuned parameters and score

print("Tuned Logistic Regression Parameters:
{}".format(logreg_cv.best_params_))

print("Best score is {}".format
```

OUTPUT :

```
Tuned Logistic Regression Parameters: {'C': 3.7275937203149381}
Best score is 0.7708333333333334
```

CODING : 2

```
import numpy as np
from sklearn.model_selection import cross_val_score
from catboost import CatBoostRegressor

# set output float precision
np.set_printoptions(precision=3)
# init model
model = CatBoostRegressor(random_seed=0, verbose=False)
# calculate mae on folds
mae = cross_val_score(model, data[best_features], data['target'],
   cv=5, scoring='neg_mean_absolute_error', n_jobs=8)
# print the results
print('folds: {}'.format(abs(mae)))
print('total: {:.3f}'.format(np.mean(abs(mae))))
```

OUTPUT :

```
folds: [1.982 2.333 2.379 1.266 2.362]
total: 2.064
```

FEATURE ENGINEERING TO IMPROVE PREDICTION MODEL'S PERFORMANCE:

Here are some key points to consider when performing feature engineering to improve a prediction model's performance:

1. Domain Knowledge: Understand the domain of your problem and the significance of each feature. This can help you identify which features are likely to be important and guide your feature engineering efforts.

2. Handling Missing Data: Deal with missing values in your dataset. You can choose to impute missing values with means, medians, or use more advanced techniques like interpolation or predictive modeling.

3. Feature Scaling: Scale numerical features to have the same range or distribution. Common techniques include standardization (mean = 0, variance = 1) or min-max scaling (scaling values to a specific range).

4. Feature Transformation: Transform features to make them more suitable for modeling. This could involve logarithmic transformations, square root transformations, or other mathematical operations to make the data more linear or normal.

5. Feature Selection: Identify and select the most relevant features for your model. Techniques like feature importance from tree-based models, correlation analysis, or forward/backward feature selection can help.

6. Creating Interaction Features: Sometimes, the interaction between two or more features can provide valuable information. Creating new features based on interactions can be beneficial.

7. Time-Series Features: For time-series data, create time-based features like rolling averages, lags, or seasonal indicators to capture temporal patterns.

8. Feature Engineering Iteration: Feature engineering is often an iterative process. You may need to try different combinations of feature engineering techniques and evaluate their impact on model performance.

9. Regularization: If your model overfits the data, consider using regularization techniques like L1 (Lasso) or L2 (Ridge) regularization to penalize certain features and prevent overfitting.

10. Validation: Always validate the impact of feature engineering on a hold-out validation set or through cross-validation to ensure improvements in model generalization performance.

CONCLUSION :

Hyperparameter tuning is a critical step in the model development process. By systematically adjusting hyperparameters like learning rates, batch sizes, and regularization strengths, we can optimize a model's performance. This process typically involves techniques such as grid search or random search to find the best combination of hyperparameters. Effective hyperparameter tuning can lead to improved model accuracy, convergence speed, and generalization ability, making it an essential part of building high-performing machine learning models.