SMARTBRIDGE
Let's Bridge the Gap
a Veranda Enterprise

Smart Internz

Model Optimization and Tuning Phase Template

Date	15 july 2024
Team ID	team-740063
Project Title	Predicting the energy output of wind turbine based on weather condition
Maximum Marks	10 Marks

Model Optimization and Tuning Phase

The model optimization and tuning phase in predicting the energy output of wind turbines based on weather conditions is crucial for improving the accuracy and reliability of predictions.and involves refining neural network models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation (8 Marks):

<table>
<tr><th>Model</th><th>Tuned Hyperparameters</th></tr>
<tr><td>Neural Network
(Multi-layer
Perceptron)</td><td>
☐ hidden_layer_sizes: Number of neurons in each hidden layer.
☐ activation: Activation function for the hidden layers.
☐ solver: Optimization algorithm to use.
☐ learning_rate_init: Initial learning rate for the optimizer
<pre><code>from sklearn.neural_network import
MLPRegressor
from sklearn.model_selection import
GridSearchCV

param_grid = {
    'hidden_layer_sizes': [(50,),
(100,), (50, 50)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam', 'lbfgs'],
    'learning_rate_init': [0.001,
0.01, 0.1]
}

mlp = MLPRegressor(random_state=42)
grid_search =
GridSearchCV(estimator=mlp,
param_grid=param_grid, cv=3,
scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

print("Best parameters found: ",
grid_search.best_params_)</code></pre>
</td></tr>
</table>

Gradient boosting Regressor

- n_estimators: Number of boosting stages to be run.
- learning_rate: Step size shrinkage.
- max_depth: Maximum depth of the individual trees.
- subsample: Fraction of samples used for fitting the individual trees.

```
from sklearn.ensemble import
GradientBoostingRegressor
from sklearn.model_selection import
RandomizedSearchCV

param_dist = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.05, 0.1,
0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 0.9, 1.0]
}

gb =
GradientBoostingRegressor(random_sta
te=42)
randomized_search =
RandomizedSearchCV(estimator=gb,
param_distributions=param_dist,
n_iter=10, cv=3,
scoring='neg_mean_squared_error',
random_state=42)
randomized_search.fit(X_train,
y_train)

print("Best parameters found: ",
randomized_search.best_params_)
```

Random Forest Regressor

- n_estimators: Number of trees in the forest.
- max_depth: Maximum depth of the trees.
- min_samples_split: Minimum number of samples required to split an internal node.
- min_samples_leaf: Minimum number of samples required to be at a leaf node

```
from sklearn.ensemble import
RandomForestRegressor
from sklearn.model_selection import
GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf =
RandomForestRegressor(random_state=4
2)
grid_search =
GridSearchCV(estimator=rf,
param_grid=param_grid, cv=3,
scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

print("Best parameters found: ",
grid_search.best_params_)
```

Final Model Selection Justification (2 Marks):

Final Model	**Reasoning**
Gradient Boosting regressor	- The Gradient Boosting Regressor was chosen as the final model due to its superior performance in terms of predictive accuracy and robustness during the model optimization phase. - It effectively handles non-linearity and complex relationships between weather variables (such as wind speed, temperature, humidity) and wind turbine energy output. - Hyperparameter tuning using RandomizedSearchCV revealed optimal settings that minimized mean squared error and generalized well across different cross-validation folds. - Compared to other models tested (such as Random Forest and Neural Network), it consistently demonstrated better performance metrics on both training and validation datasets.