

Model Optimization and Tuning Phase

Template

| | |
|---------------|--|
| Date | 11 July 2024 |
| Team ID | SWTID1720162737 |
| Project Title | Predicting Compressive Strength Of Concrete Using Machine Learning |
| Maximum Marks | 10 Marks |

Model Optimization and Tuning Phase:

The Model Optimization and Tuning Phase involves refining machine learning 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 (6 Marks):

| Model | Tuned Hyperparameters | Optimal Values |
|-------------------|-----------------------|--|
| Linear Regression | --- | <pre>acc=r2_score(y_test,pred) print('accuracy of linear regression Regression:',acc*100) accuracy of linear regression Regression: 58.61758560675364</pre> |
| Ridge Regression | --- | <pre>acc=r2_score(y_test,pred) print('accuracy of Ridge regression Regression:',acc*100) accuracy of Ridge regression Regression: 58.61760529691513</pre> |
| Lasso Regression | --- | <pre>acc=r2_score(y_test,pred) print('accuracy of lasso regression Regression:',acc*100) accuracy of lasso regression Regression: 58.78727632129104</pre> |

| | | |
|------------------------------|--|---|
| Random Forest Regression | <pre># Initialize the RandomForestRegressor rfr = RandomForestRegressor() # Define the parameter grid param_grid = { 'n_estimators': [100, 200, 300, 400, 500], 'max_depth': [None, 10, 20, 30, 40, 50], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootstrap': [True, False] } # Initialize GridSearchCV grid_search = GridSearchCV(estimator=rfr, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2) # Fit the model grid_search.fit(x_train, y_train)</pre> <p>GridSearchCV got the higher accuracy.</p> | <pre>print(f'Optimal Hyperparameters:{best_params}') acc=r2_score(y_test,pred) print('accuracy of RandomForestRegression:',acc*100) Optimal Hyperparameters: {'bootstrap': True, 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200} accuracy of RandomForestRegression: 93.22882639139073</pre> |
| Decision Tree Regression | --- | <pre>acc=r2_score(y_test,pred) print('accuracy of Decision Tree Regression:',acc*100) accuracy of Decision Tree Regression: 88.49267192424941</pre> |
| Gradient Boosting regression | <pre>gb = GradientBoostingRegressor() # Define the parameter distribution param_dist = { 'n_estimators': randint(100, 500), 'learning_rate': uniform(0.01, 0.2), 'max_depth': randint(1, 10), 'min_samples_split': randint(2, 20), 'min_samples_leaf': randint(1, 20), 'subsample': uniform(0.5, 0.8), 'max_features': ['auto', 'sqrt', 'log2', None] } # Initialize RandomizedSearchCV random_search = RandomizedSearchCV(estimator=gb, param_distributions=param_dist, n_iter=100, cv=5, verbose=2, random_state=42, n_jobs=-1) # Fit the model random_search.fit(x_train, y_train)</pre> <p>RandomizedSearchCV got the higher accuracy.</p> | <pre>print(f'Optimal Hyperparameters:{best_params}') acc=r2_score(y_test,pred) print('accuracy of XGBoost Regression:',acc*100) Optimal Hyperparameters: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 500} accuracy of XGBoost Regression: 93.80577042460435</pre> |
| XGBoost Regression | <pre>#importing and building XGBoost Regression using RandomizedSearchCV import xgboost as xgb xg=xgb.XGBRegressor() # Define the parameter distribution param = { 'n_estimators': randint(100, 500), 'learning_rate': uniform(0.01, 0.2), 'max_depth': randint(3, 10), 'min_child_weight': randint(1, 10), 'subsample': uniform(0.5, 0.8), 'colsample_bytree': uniform(0.5, 0.8) } # Initialize RandomizedSearchCV random_search = RandomizedSearchCV(estimator=xg, param_distributions=param, n_iter=100, cv=5, verbose=2, random_state=42, n_jobs=-1) random_search.fit(x_train,y_train)</pre> <p>RandomizedSearchCV got the higher accuracy.</p> | <pre>print(f'Optimal Hyperparameters:{best_params}') acc=r2_score(y_test,pred) print('accuracy of XGBoost Regression:',acc*100) Optimal Hyperparameters: {'colsample_bytree': 0.954660201039391, 'learning_rate': 0.061755996320003385, 'max_depth': 6, 'min_child_weight': 2, 'n_estimators': 489, 'subsample': 0.6039708314340944} accuracy of XGBoost Regression: 94.18826209575943</pre> |

Performance Metrics Comparison Report (2 Marks):

| Model | Baseline Metric | Optimized Metric |
|-------|-----------------|------------------|
|-------|-----------------|------------------|

| | | |
|------------------------------|---|---|
| Linear Regression | <pre>acc=r2_score(y_test,pred) print('accuracy of linear regression Regression:',acc*100) accuracy of linear regression Regression: 58.61758560675364 print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) MSE: 110.18594180669501 MAE: 8.261594213211119 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) RMSE: 10.49694916662432</pre> | <pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) MSE: 110.18594180669501 MAE: 8.261594213211119 RMSE: 10.49694916662432</pre> |
| Ridge Regression | <pre>acc=r2_score(y_test,pred) print('accuracy of Ridge regression Regression:',acc*100) accuracy of Ridge regression Regression: 58.61760529691513 print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) MSE: 110.18589937913565 MAE: 8.261589878127149 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) RMSE: 10.49694666934798</pre> | <pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) MSE: 110.18589937913565 MAE: 8.261589878127149 RMSE: 10.49694666934798</pre> |
| Lasso Regression | <pre>acc=r2_score(y_test,pred) print('accuracy of lasso regression Regression:',acc*100) accuracy of lasso regression Regression: 58.78727632129104 print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) MSE: 109.73411869605009 MAE: 8.244898906700172 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) RMSE: 10.475405419173526</pre> | <pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) MSE: 109.73411869605009 MAE: 8.244898906700172 RMSE: 10.475405419173526</pre> |
| RandomForest Regression | <pre>acc=r2_score(y_test,pred) print('accuracy of RandomForest Regression:',acc*100) accuracy of RandomForest Regression: 88.49267192424941 print("MSE: ",metrics.mean_squared_error(pred,y_test)) print("RMSE: ",np.sqrt(metrics.mean_squared_error(pred,y_test))) MSE: 21.511922600580977 RMSE: 4.6380947166461555</pre> | <pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) MSE: 23.587089288556324 MAE: 3.514265078126178 RMSE: 4.856654124863775</pre> |
| Decision Tree Regression | <pre>acc=r2_score(y_test,pred) print('accuracy of Decision Tree Regression:',acc*100) accuracy of Decision Tree Regression: 88.49267192424941 print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) MSE: 30.63972463414634 MAE: 3.6327804878048777 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) RMSE: 5.5353161277515435</pre> | <pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) MSE: 30.63972463414634 MAE: 3.6327804878048777 RMSE: 5.5353161277515435</pre> |
| Gradient Boosting regression | <pre>score=r2_score(y_pred,y_test) print("accuracy of GradientBoosting Regression:: ",score*100) accuracy of GradientBoosting Regression:: 88.26767558265551 print("MSE: ",metrics.mean_squared_error(pred,y_test)) print("RMSE: ",np.sqrt(metrics.mean_squared_error(pred,y_test))) MSE: 30.63972463414634 RMSE: 5.5353161277515435</pre> | <pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) MSE: 25.861465265637968 MAE: 3.6224784271501735 RMSE: 5.085416921515675</pre> |

| | | |
|------------------------|--|--|
| XG Boost Regression | <pre>score=r2_score(pred,y_test) print("accuracy of XGBoost Regression:: ",score*100) accuracy of XGBoost Regression:: 92.62911257081528 print("MSE: ",metrics.mean_squared_error(pred,y_test)) print("RMSE: ",np.sqrt(metrics.mean_squared_error(pred,y_test))) MSE: 17.96478280626198 RMSE: 4.238488268977747</pre> | <pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) MSE: 15.474491372763513 MAE: 2.4343691059205588 RMSE: 3.933763004142918</pre> |
|------------------------|--|--|

Final Model Selection Justification (2 Marks):

| Final Model | Reasoning |
|-------------------------|---|
| XG Boost Regression. | ‘XG Boost Regression’ the best performance and generalizability on unseen data, considering factors beyond just raw accuracy. |