

## Model Development Phase Template

Date	10 July 2024
Team ID	SWTID1720162737
Project Title	Predicting Compressive Strength Of Concrete Using Machine Learning
Maximum Marks	4 Marks

### Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include classification reports, accuracy, and confusion matrices for multiple models, presented through respective screenshots.

#### Initial Model Training Code:

Importing and building linear regression model:

```
#importing and building linear regression model
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
# Training the Model
lr.fit(x_train,y_train)
pred = lr.predict(x_test)
```

```
acc=r2_score(y_test,pred)
print('accuracy of linear regression Regression:',acc*100)
```

```
accuracy of linear regression Regression: 58.61758560675364
```

Importing and uilding Ridge Regression:

```
#importing and building Ridge Regression
from sklearn.linear_model import Ridge
r=Ridge()
# Ridge Training
r.fit(x_train,y_train)
pred=r.predict(x_test)
```

```
acc=r2_score(y_test,pred)
print('accuracy of Ridge regression Regression:',acc*100)
```

```
accuracy of Ridge regression Regression: 58.61760529691513
```

Importing and Building Lasso Regression:

```
#importing and building Lasso Regression
from sklearn.linear_model import Lasso
l=Lasso()
#lasso training
l.fit(x_train,y_train)
pred=l.predict(x_test)
```

```
acc=r2_score(y_test,pred)
print('accuracy of lasso regression Regression:',acc*100)
```

```
accuracy of lasso regression Regression: 58.78727632129104
```

Importing and Building Random Forest Regression using hyperparameter GridSearchCV:

```
# 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)
```

Fitting 5 folds for each of 540 candidates, totalling 2700 fits

```
GridSearchCV
  best_estimator_: RandomForestRegressor
    RandomForestRegressor
```

```
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
```

### Importing and Building Decision Tree Regression:

```
#importing and building Decision Tree Regression
from sklearn.tree import DecisionTreeRegressor
df=DecisionTreeRegressor(criterion='squared_error',random_state=0)
df.fit(x_train,y_train)
pred=df.predict(x_test)
```

```
acc=r2_score(y_test,pred)
print('accuracy of Decision Tree Regression:',acc*100)
```

```
accuracy of Decision Tree Regression: 88.49267192424941
```

### Importing and Building Gradient Boosting Regression using GridSearchCV:

```
from sklearn.ensemble import GradientBoostingRegressor
gb = GradientBoostingRegressor()

# Define the parameter distribution
param_dist = {
    'n_estimators': [100, 200, 300, 400, 500],
    'learning_rate': [0.1, 0.05, 0.01, 0.005],
    'max_depth': [3, 4, 5, 6],
    # ... other hyperparameters
}

grid_search = GridSearchCV(estimator=gb, param_grid=param_dist, cv=5, verbose=2, n_jobs=-1)
grid_search.fit(x_train,y_train)
```

Fitting 5 folds for each of 80 candidates, totalling 400 fits

```
> GridSearchCV ⓘ ⓘ
  > best_estimator_: GradientBoostingRegressor
    > GradientBoostingRegressor ⓘ
```

```
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

## 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, 0.5),
    'colsample_bytree': uniform(0.5, 0.5)
}

# 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)
```

Fitting 5 folds for each of 100 candidates, totalling 500 fits

```
> RandomizedSearchCV ⓘ ⓘ
  > best_estimator_: XGBRegressor
    > XGBRegressor
```

```
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

## Model Validation and Evaluation Report:

Model	Mean Squared Error Mean Absolute Error	Accuracy (R2 Score)	RMSE
Linear Regression	<pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) MSE: 110.18594180669501 MAE: 8.261594213211119</pre>	58.61 %	<pre>print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) RMSE: 4.246070061545925</pre>
Ridge Regression	<pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) MSE: 110.1858937913565 MAE: 8.261589878127149</pre>	58.62%	<pre>print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) RMSE: 10.49694666934798</pre>
Lasso Regression	<pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) MSE: 109.73411869605009 MAE: 8.244898906700172</pre>	58.78 %	<pre>print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) RMSE: 10.475405419173526</pre>
Random Forest Regression	<pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) MSE: 21.61696764611182 MAE: 3.208414090592337</pre>	93.23%	<pre>print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) RMSE: 4.856654124863775</pre>
Decision Tree Regression	<pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) MSE: 30.63972463414634 MAE: 3.6327804878048777</pre>	88.49%	<pre>print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) RMSE: 5.5353161277515435</pre>
Gradient Boosting regression	<pre>print('MSE:', metrics.mean_squared_error(y_test,pred)) print('MAE:', metrics.mean_absolute_error(y_test,pred)) MSE: 16.666770682562888 MAE: 2.565047486791529</pre>	93.8%	<pre>print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) RMSE: 4.096242452448049</pre>
XGBoost Regression	<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: 18.029110967556615 MAE: 2.623841562457201</pre>	94.18%	<pre>print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pred))) RMSE: 3.933763004142918</pre>