Low Level Design (LLD)

Concrete Compressive strength Prediction

Written by	Sanket Nivrutti Jagtap
Document version	0.2
Last revised date	01–Nov-2024

Document Version Control

Change record:

Version	Date	Author	Comments
0.1	30-Oct-24	Sanket Nivrutti Jagtap	Introduction and architecture defined
0.2	01-Nov-24	Sanket Nivrutti Jagtap	Architecture and architecture description updated. Unit test cases defined and updated

Contents

1. Introduction

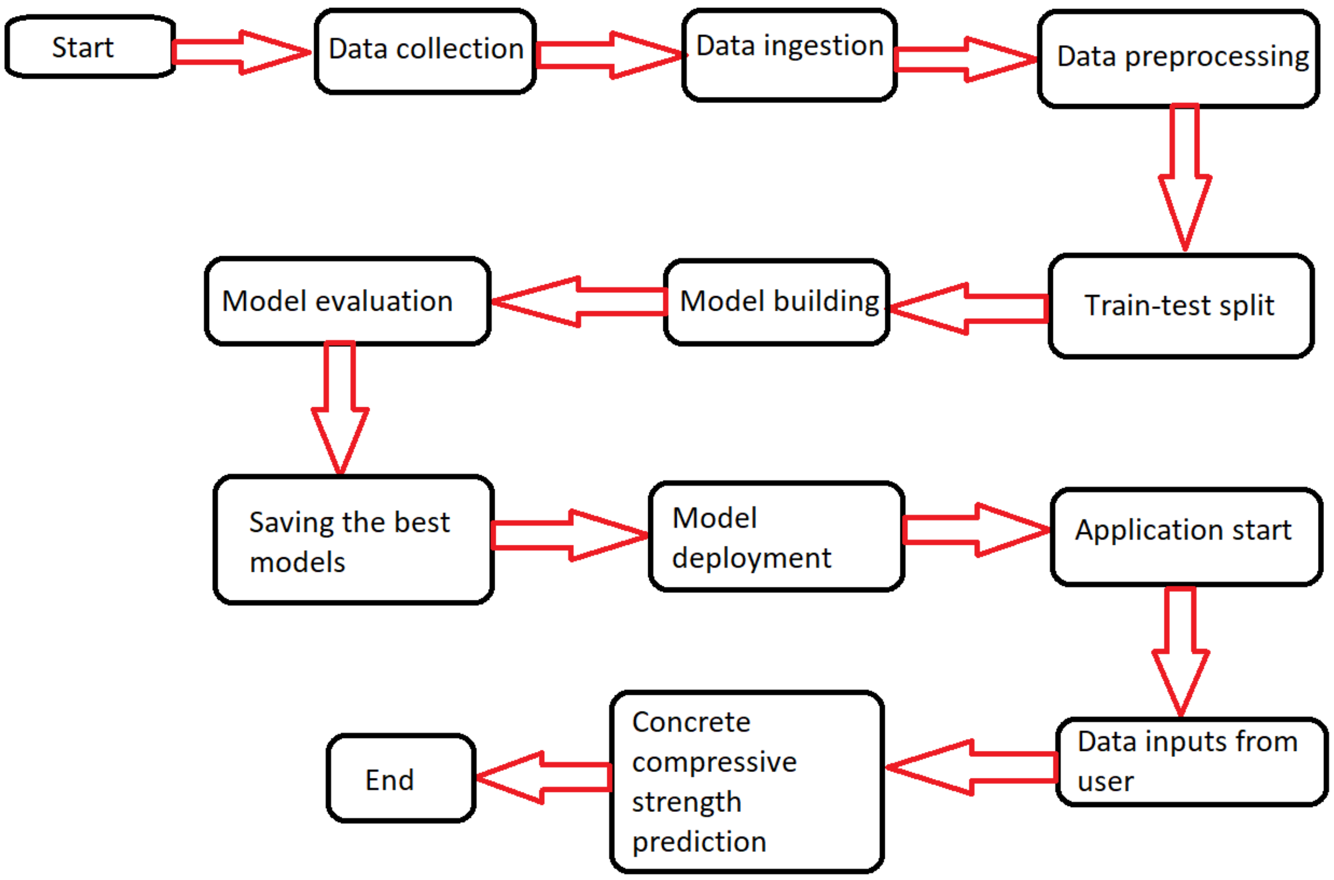
1.1 What is Low-Level design document?

The goal of LLD or a Low-level design document is to give an internal logical design of the actual program code for the Concrete Compressive Strength Prediction System. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

1.2 Scope

Low-level design (LLD) is a component level design process that follows a step-by-step refinement process. This process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then defined during data design work.

2. Architecture



3. Architecture Description

3.1 Data Collection

For training and testing the model, we used the public data set available in Kaggle, “Concrete Compressive Strength Data Set” by Ahiale Darlington.
URL - https://www.kaggle.com/elikplim/concrete-compressive-strength-data-set

Data dictionary is as follows:
The actual concrete compressive strength (MPa) i.e., mega pascals for a given mixture under a specific age (days) was determined from laboratory. Data is in raw form (not scaled).

Summary Statistics:
Number of instances (observations): 1030 Number of Attributes: 9 Attribute breakdown: 8 quantitative input variables, and 1 quantitative output variable.
Missing Attribute Values: None

Variable Information:
Given is the variable name, variable type, the measurement unit and a brief description. The concrete compressive strength is the regression problem. The order of this listing corresponds to the order of numerals along the rows of the database.

Name	Data Type	Measurement	Description
Cement	Quantitative	kg in a m3 mixture	Input variable
Blast Furnace Slag	Quantitative	kg in a m3 mixture	Input variable
Fly Ash	Quantitative	kg in a m3 mixture	Input variable
Water	Quantitative	kg in a m3 mixture	Input variable
Superplasticizer	Quantitative	kg in a m3 mixture	Input variable
Coarse Aggregate	Quantitative	kg in a m3 mixture	Input variable
Fine Aggregate	Quantitative	kg in a m3 mixture	Input variable
Age	Quantitative	Days (1~365)	Input variable
Concrete Compressive Strength	Quantitative	megapascals (MPa)	Output variable

3.2 Data ingestion

For loading the dataset into the coding environment, I created a module called “data_ingestion” containing “data_loader.py” file. In that file, I created a class called “DataLoad” and it takes the dataset file path as input. Inside that class,

- I created a method “fetch_data” whose duty is to load the dataset as a pandas data-frame into the coding environment and records logs in the “development_logs.log” file.

3.3 Data pre-processing

For data pre-processing, I created a module called “data_preprocessing” containing “data_preprocessing.py” file. In that file, I created a class called DataPreprocessor which takes the dataset in the form of pandas data-frame as an input. Inside that class, I created

- A method “rem_outliers”, takes a particular column name as an input and removes outliers using Interquartile range (IQR) method in that column and records logs.
- A method “data_split”, which takes the percentage of test data i.e., test size as an input and splits the data-frame into training and testing data-frames respectively using the “train_test_split” method from scikit learn. Logs will be updated
- A method “feature_scaling”, which takes the training and testing data-frames as inputs and scales the features in both using the StandardScaler from scikit learn library. Logs will be updated. This method is used only for the Linear regression models.
- A method “splitting_as_x_y”, which takes the training data-frame, testing data-frame and the target column’s name as inputs, splits both the training and testing data-frames into the dataframes containing independent and dependent features respectively.

  For example:
  If we pass “df_train”,” df_test” and “price” as inputs to this method, it returns the data-frames, “x_train” without “price” column, “y_train” with only the “price” column, “x_test” without “price” column and “y_test” with only the “price” column. Logs will be updated.

3.4 Model building

For model building, I created a module called “algorithms” containing two files, “linear_models.py” and “tree_models.py”.

a) “linear_models.py”

Inside this file, I created two classes- “LinearRegressionWithFeatureSelection” and “Lasso”. Both classes take the x_train, y_train, x_test and y_test data-frames respectively as their inputs.

The former class uses two methods and logs will be updated accordingly–

“backward_elimination_approach”: -Builds a linear regression model on all the features, eliminates each one with respect to its p value, if it is above 0.5. Then the left-over features are the relevant features, which are used to build a Linear regression model. It returns the linear regression model, its predictions on both the training and testing data-frames and the relevant features in the form of python list.

“rfe_approach”: - Uses Recursive feature elimination or RFE technique from scikit learn library, ultimately selects the most relevant features in the dataset. Then using these features a Linear regression model is built. It returns the linear regression model, its predictions on both the training and testing data-frames and the relevant features.

The latter class “Lasso” uses a method called, “lassocv”. This method uses LassoCV algorithm imported from the sci-kit learn library to build a linear regression model. It does a cross validation with various learning rates, ultimately finds the most relevant features and builds a regression model on them. It returns the final model, its predictions on both the training and testing data-frames and displays the importance of each feature in the console. Logs will be updated accordingly

b) “tree_models.py”

Inside this file, I created a class called “TreeModelsReg” which takes the x_train, y_train, x_test and y_test data-frames respectively as the inputs.

This class contains the following methods, and each method carries out logging operation:

“decision_tree_regressor”: This method builds a model using DecisionTreeRegressor algorithm imported from the scikit learn library, by considering the best hyper parameters, after performing the randomized search cross validation technique. It returns the model and displays the importance of each feature in the console.

“decision_tree_regressor_post_pruning”: This method implements the post pruning technique to tackle over-fitting problem in the decision tree regressor. While doing so, we found out the optimum cost complexity pruning or ccp_alpha parameter as 0.8 in the “EDA + Model building. ipynb” jupyter notebook using visualization. It returns the final model.

“random_forest_regressor”: - This method builds a model using RandomForestRegressor algorithm imported from the scikit learn library, by considering the best hyperparameters, after performing the randomized search cross validation technique. It returns the model and displays the importance of each feature in the console.

“adaboost_regressor”: - This method builds a model using AdaBoostRegressor algorithm imported from the scikit learn library, by considering the best hyperparameters, after performing the randomized search cross validation technique. It returns the model and displays the importance of each feature in the console.

“gradientboosting_regressor”: - This method builds a model using GradientBoostingRegressor algorithm imported from the scikit learn library, by considering the best hyperparameters, after performing the randomized search cross validation technique. It returns the model and displays the importance of each feature in the console.

"xgb_regressor": - This method builds a model using the XGBRegressor algorithm imported from the xgboost library, by considering the best hyperparameters, after performing the randomized search cross validation technique. It returns the model and displays the importance of each feature in the console.

"model_predict": - This method is created to make predictions on all the above defined tree model methods. It takes the model as an input and returns the prediction.

The important features of each model with respect to their model's name, will be stored in the "relevant_features_by_models.csv" file in the "results" directory.

3.5 Model evaluation

For model evaluation, I created a module called "evaluation" containing a file

"evaluation py". Inside that file, I created a class called "Metrics" with the following methods and logs will be updated by each one of them accordingly.

"r2_score": - This method calculates the r2_score of a model, by taking both the true and the predicted values of the target variable and returns the result.

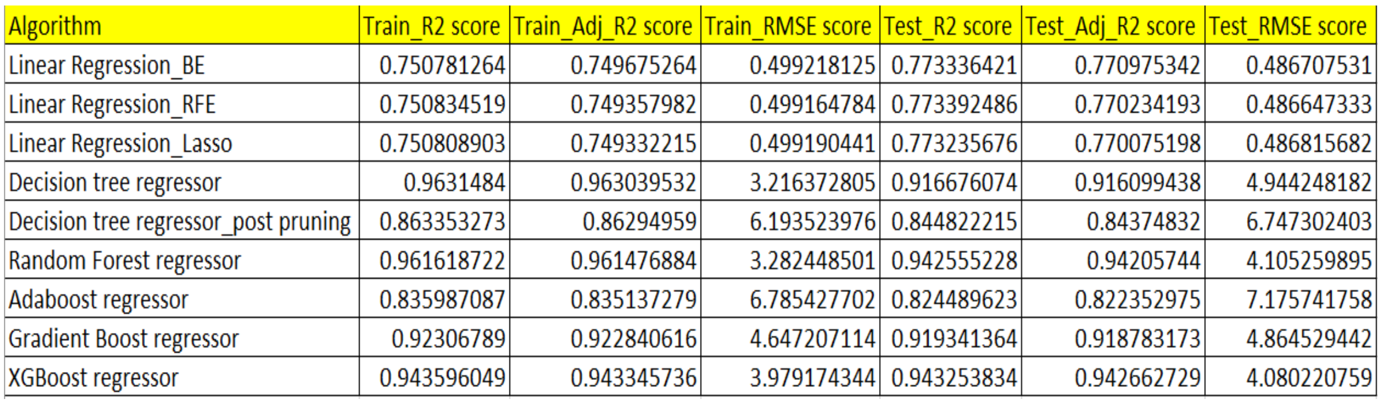
"adj_r2_score": - This method calculates the adjusted r2 score of a model, by taking the data-frame containing the predictor features, the true and the predicted values of the target variable and returns the result.

"rmse_score": - This method calculates the root mean square error of a model on the given dataset, by taking both the true and the predicted values of the target variable and returns the result.

All these results with respect to their algorithm name will be stored in a "performance of algorithms.csv" file in the results directory.

3.6 Saving the best models

Based on the below results, I saved the “XGBoost regressor” model and the “Random Forest regressor” model into the “models” directory using the joblib library, as these two are the best among all.



Algorithm	Train_R2 score	Train_Adj_R2 score	Train_RMSE score	Test_R2 score	Test_Adj_R2 score	Test_RMSE score
Linear Regression_BE	0.750781264	0.749675264	0.499218125	0.773336421	0.770975342	0.486707531
Linear Regression_RFE	0.750834519	0.749357982	0.499164784	0.773392486	0.770234193	0.486647333
Linear Regression_Lasso	0.750808903	0.749332215	0.499190441	0.773235676	0.770075198	0.486815682
Decision tree regressor	0.9631484	0.963039532	3.216372805	0.916676074	0.916099438	4.944248182
Decision tree regressor_post pruning	0.863353273	0.86294959	6.193523976	0.844822215	0.84374832	6.747302403
Random Forest regressor	0.961618722	0.961476884	3.282448501	0.942555228	0.94205744	4.105259895
Adaboost regressor	0.835987087	0.835137279	6.785427702	0.824489623	0.822352975	7.175741758
Gradient Boost regressor	0.92306789	0.922840616	4.647207114	0.919341364	0.918783173	4.864529442
XGBoost regressor	0.943596049	0.943345736	3.979174344	0.943253834	0.942662729	4.080220759

All the development part of the code runs in the “main.py” file using the above-mentioned modules

3.7 Model deployment

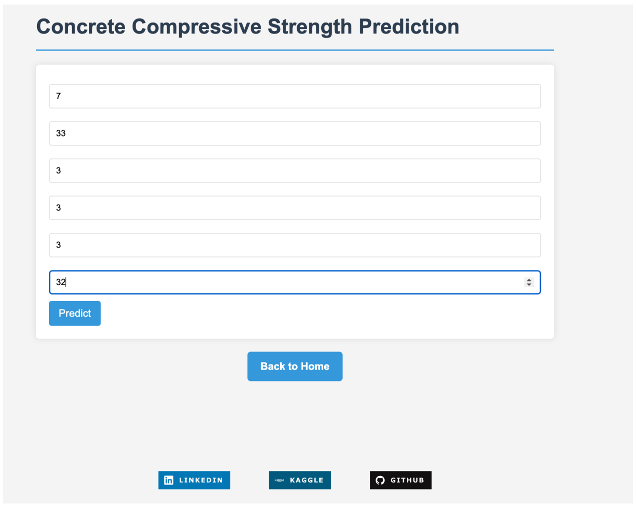
Deployed the “XGBoost regressor” model in the web application using Flask a micro web framework in python. The deployment part of the code runs in the “app.py” file, connecting with the web page designed using HTML with CSS styles.

Then, deployed on web using the GitHub a version control system, Gunicorn version 20.1.0 and the Koyeb as a platform as a service (PaaS).

Web application URL - **https://eventual-elnore-sanketjagtap-7e11777b.koyeb.app/**

Here’s how the web application looks like: -

Users or engineers can input the details about the concrete specimen and once they hit "Predict" button, the predicted concrete compressive strength will be displayed as below


Concrete Compressive Strength Prediction

7
33
3
3
3
32

4. Unit Test Cases

Test Case Description	Pre-requisite	Expected Result
Verify whether the Application URL is accessible to the user	1. Application URL should be defined	Application URL should be accessible to the user
Verify whether the Application loads completely for the user when the URL is accessed.	1. Application URL is accessible 2. Application is deployed	The application should load completely for the user when the URL is accessed
Verify whether user can edit all the input fields	1. Application URL is accessible 2. Application loads completely for the user. 3. All the input fields loaded	User should be able to edit all the input fields
Verify whether user gets "Predict" button to make predictions on the given inputs	1. Application URL is accessible 2. Application loads completely for the user. 3. All the input fields loaded	User should get a "Predict" button to make predictions on the given inputs.
Verify whether user is presented with recommended results on clicking the "Predict" button	1. Application URL is accessible 2. Application loads completely for the user 3. All the input fields loaded	Users should be presented with recommended results on clicking the "Predict" button.