Predicting the compressive strength of concrete

1.Introduction

Concrete is one of the most important materials in Civil Engineering. Knowing the compressive strength of concrete is very important when constructing a building or a bridge. The Compressive Strength of Concrete is a highly nonlinear function of ingredients used in making it and their characteristics. Thus, using Machine Learning to predict the Strength could be useful in generating a combination of ingredients which result in high Strength.

Description and Project Overview This project will be based on a dataset obtained from the UCI Repository. The dataset consists of 1030 observations under 9 attributes. The attributes consist of 8 quantitative inputs and 1 quantitative output. The dataset does not contain any missing values. The dataset is focused on the compressive strength of a concrete. The attributes include factors that affect concrete strength such as cement, water, aggregate (coarse and fine), and fly ash etc… The objective of this project is trying to predict the concrete compressive strength based important predictors. The study will consist of evaluating the impact of different factors such as cement, water, age, fly ash, and or additives. We will evaluate the components that are highly correlated with concrete compressive strength and other components that are less influential and can be neglected through visualization or correlation matrix. In this study, we will use different machine learning techniques to predict the concrete compressive strength. Different modeling techniques will be used for the prediction. The modeling technique will include multiple linear regression, decision tree, and random forest, etc. A comparative analysis will be performed to identify the best model for our prediction in terms of accuracy. The best model will be helpful for civil engineers in choosing the appropriate concrete for bridges, houses construction.

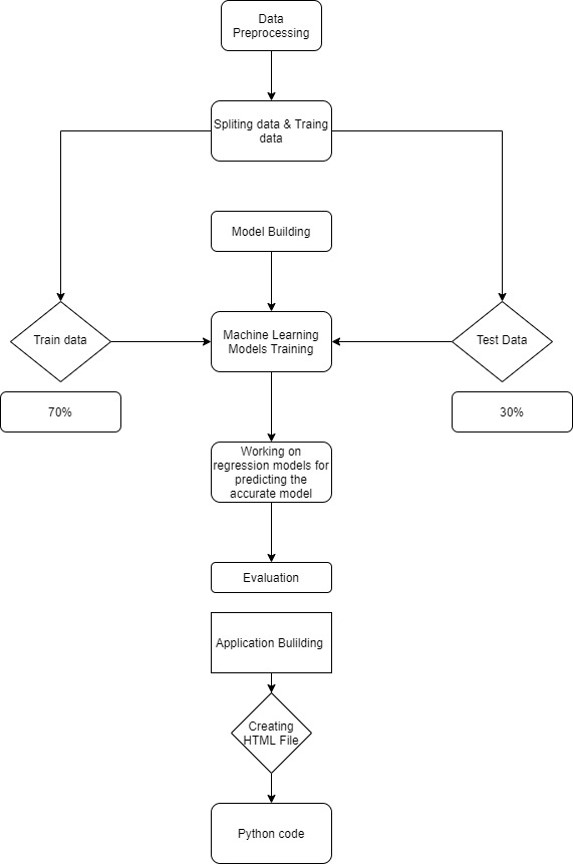
2. EXISTING SOCIETAL ISSUE:

In earlier days, the concrete strength is measure through other traditional methods like using drill holes, weight spring, or using sensors. But that requires a significant destruction of test sample and thereby increasing the cost. The recommended wait time for testing the cylinder is 28 days to ensure correct results. This consumes a lot of time and requires a lot of labour to prepare different prototypes and test them. Also, this method is prone to human error and one small mistake can cause the wait time to drastically increase.

PROPOSED SOLUTION:

The focus of this project is the application of machine learning process,Artificial nueral networks and their suitability to model concrete compressive strength compared with early models obtained from the literature and compared with some conventional approaches and also a recoomendation system is developed by applying various ML methods,Deep nueral network methods to predict the concrete strength from its components accurately and then looking for the optimal combination of components which increases the strength.

ARCHITECTURE DIAGRAM:



DATA DESCRIPTION:

Data is obtained from UCI Machine Learning Repository and this dataset is used for all ML Algorithms. <https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength>

Number of instances - 1030 Number of Attributes - 9 Attribute breakdown - 8 quantitative inputs, 1 quantitative output

Attribute information:

Inputs: Cement, Blast Furnace Slag, Fly Ash, Water, Superplasticizer, Coarse Aggregate, Fine Aggregate, All above features measured in kg/$m^3$, Age (in days),

Output: Concrete Compressive Strength (Mpa)

Dataset used for Deep Nueral Network is <https://www.kaggle.com/maajdl/yeh-concret-data>

MODELLING AND EVALUATION:

ML Algorithms used: Linear regression, Lasso regression, Ridge regression, Decision Trees, Random Forests, Deep Nueral Nework

Metric - Since the target variable is a continuous variable, regression evaluation metric RMSE (Root Mean Squared Error) and R2 Score (Coefficient of Determination) have been used. And a recommendation system is developed as which algorithm is best choice for predicting accurate concrete strength.

Algorithms used

Linear regression

Lasso regression

Ridge regression

Decision Trees

Random Forests

Metric - Since the target variable is a continuous variable, regression evaluation metric RMSE (Root Mean Squared Error) and R2 Score (Coefficient of Determination) have been used.

The data contains 1030 examples and the following features:

Input Variable: Cement (kg in a m^3 mixture)

Input Variable: Blast Furnace Slag (kg in a m^3 mixture)

Input Variable: Fly Ash (kg in a m^3 mixture)

Input Variable: Water (kg in a m^3 mixture)

Input Variable: Superplasticizer (kg in a m^3 mixture)

Input Variable: Coarse Aggregate (kg in a m^3 mixture)

Input Variable: Fine Aggregate (kg in a m^3 mixture)

Input Variable: Age (days)

Output Variable: Concrete compressive strength (MPa)

Exploratory Data Analysis

The first step in a Data Science project is to understand the data and gain insights from the data before doing any modelling. This includes checking for any missing values, plotting the features with respect to the target variable, observing the distributions of all the features and so on. Let us import the data and start analysing.

Let us check the correlations between the input features, this will give an idea about how each variable is affecting all other variables. This can be done by calculating Pearson correlations between the features as shown in the code below.

corr = data.corr() sns.heatmap(corr, annot=True, cmap='Blues')

conclusion

Analysed the Compressive Strength and used Machine Learning to Predict the Compressive Strength of Concrete. We have used Linear Regression and its variations, Decision Trees and Random Forests to make predictions and compared their performance. Random Forest Regressor has the lowest RMSE and is a good choice for this problem. Also, we can further improve the performance of the algorithm by tuning the hyperparameters by performing a grid search or random search.

NN approaches combine the complexity of many statistical techniques with machine learning techniques and attributed as a black-box which allows NN to be applied in all engineering disciplines. It comes out as the best possible model for the prediction of compressive strength of concrete. It has predicted with high accuracy for all the curing ages, that is, 28, 56, and 91 days.

SOURCE CODE

#!/usr/bin/env python

# coding: utf-8

# In[1]:

#importing libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import itertools

get\_ipython().run\_line\_magic('matplotlib', 'inline')

# In[2]:

data=pd.read\_csv("F:\project\Concrete.csv")

# In[3]:

len(data)

# In[35]:

req\_col\_names = ["cement", "slag", "flyAsh", "water", "superplasticizer",

"coarseAggregate", "fineAggregare", "age", "csMPa"]

curr\_col\_names = list(data.columns)

mapper = {}

for i, name in enumerate(curr\_col\_names):

mapper[name] = req\_col\_names[i]

data = data.rename(columns=mapper)

# In[5]:

data.head()

# In[ ]:

# checking for null values

# In[6]:

data.isna().sum()

# In[8]:

#There are no null values in the data

# In[9]:

#Checking the pairwise relations of Features

# In[10]:

sns.pairplot(data)

plt.show()

# In[11]:

#Correlation coefficients between the features.

# In[12]:

corr = data.corr()

sns.heatmap(corr, annot=True, cmap='Blues')

b, t = plt.ylim()

plt.ylim(b+0.5, t-0.5)

plt.title("Feature Correlation Heatmap")

plt.show()

# In[13]:

#There are'nt any high correlations, except between Cement and Compressive Strength of Concrete. Which should be the case for strength.

# In[19]:

ax = sns.distplot(data.csMPa)

ax.set\_title("Compressive Strength Distribution")

# In[21]:

fig, ax = plt.subplots(figsize=(10,7))

sns.scatterplot(y="csMPa", x="cement", hue="water", size="age", data=data, ax=ax, sizes=(20, 200))

ax.set\_title("csMPa vs (cement, age, water)")

ax.legend(loc="upper left", bbox\_to\_anchor=(1,1))

plt.show()

# In[22]:

#Data Preprocessing

# In[23]:

X = data.iloc[:,:-1]

y = data.iloc[:,-1]

# In[24]:

#Splitting data into Training and Test splits

# In[25]:

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=2)

# In[26]:

#Scaling

# In[27]:

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# In[28]:

#Model Building

# In[29]:

#Linear Regression

# In[30]:

# Importing models

from sklearn.linear\_model import LinearRegression, Lasso, Ridge

# Linear Regression

lr = LinearRegression()

# Lasso Regression

lasso = Lasso()

# Ridge Regression

ridge = Ridge()

# Fitting models on Training data

lr.fit(X\_train, y\_train)

lasso.fit(X\_train, y\_train)

ridge.fit(X\_train, y\_train)

# Making predictions on Test data

y\_pred\_lr = lr.predict(X\_test)

y\_pred\_lasso = lasso.predict(X\_test)

y\_pred\_ridge = ridge.predict(X\_test)

# In[31]:

#Evaluation

# In[32]:

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

print("Model\t\t\t RMSE \t\t MSE \t\t MAE \t\t R2")

print("""LinearRegression \t {:.2f} \t\t {:.2f} \t{:.2f} \t\t{:.2f}""".format(

np.sqrt(mean\_squared\_error(y\_test, y\_pred\_lr)),mean\_squared\_error(y\_test, y\_pred\_lr),

mean\_absolute\_error(y\_test, y\_pred\_lr), r2\_score(y\_test, y\_pred\_lr)))

print("""LassoRegression \t {:.2f} \t\t {:.2f} \t{:.2f} \t\t{:.2f}""".format(

np.sqrt(mean\_squared\_error(y\_test, y\_pred\_lasso)),mean\_squared\_error(y\_test, y\_pred\_lasso),

mean\_absolute\_error(y\_test, y\_pred\_lasso), r2\_score(y\_test, y\_pred\_lasso)))

print("""RidgeRegression \t {:.2f} \t\t {:.2f} \t{:.2f} \t\t{:.2f}""".format(

np.sqrt(mean\_squared\_error(y\_test, y\_pred\_ridge)),mean\_squared\_error(y\_test, y\_pred\_ridge),

mean\_absolute\_error(y\_test, y\_pred\_ridge), r2\_score(y\_test, y\_pred\_ridge)))

# In[33]:

#Plotting the coefficients

# In[36]:

coeff\_lr = lr.coef\_

coeff\_lasso = lasso.coef\_

coeff\_ridge = ridge.coef\_

labels = req\_col\_names[:-1]

x = np.arange(len(labels))

width = 0.3

fig, ax = plt.subplots(figsize=(10,6))

rects1 = ax.bar(x - 2\*(width/2), coeff\_lr, width, label='LR')

rects2 = ax.bar(x, coeff\_lasso, width, label='Lasso')

rects3 = ax.bar(x + 2\*(width/2), coeff\_ridge, width, label='Ridge')

ax.set\_ylabel('Coefficient')

ax.set\_xlabel('Features')

ax.set\_title('Feature Coefficients')

ax.set\_xticks(x)

ax.set\_xticklabels(labels, rotation=45)

ax.legend()

def autolabel(rects):

"""Attach a text label above each bar in \*rects\*, displaying its height."""

for rect in rects:

height = rect.get\_height()

ax.annotate('{:.2f}'.format(height), xy=(rect.get\_x() + rect.get\_width() / 2, height),

xytext=(0, 3), textcoords="offset points", ha='center', va='bottom')

autolabel(rects1)

autolabel(rects2)

autolabel(rects3)

fig.tight\_layout()

plt.show()

# In[37]:

#Plotting predictions

# In[38]:

fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(12,4))

ax1.scatter(y\_pred\_lr, y\_test, s=20)

ax1.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=2)

ax1.set\_ylabel("True")

ax1.set\_xlabel("Predicted")

ax1.set\_title("Linear Regression")

ax2.scatter(y\_pred\_lasso, y\_test, s=20)

ax2.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=2)

ax2.set\_ylabel("True")

ax2.set\_xlabel("Predicted")

ax2.set\_title("Lasso Regression")

ax3.scatter(y\_pred\_ridge, y\_test, s=20)

ax3.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=2)

ax3.set\_ylabel("True")

ax3.set\_xlabel("Predicted")

ax3.set\_title("Ridge Regression")

fig.suptitle("True vs Predicted")

fig.tight\_layout(rect=[0, 0.03, 1, 0.95])

# In[39]:

#Decision Trees

# In[40]:

from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor()

dtr.fit(X\_train, y\_train)

y\_pred\_dtr = dtr.predict(X\_test)

print("Model\t\t\t\t RMSE \t\t MSE \t\t MAE \t\t R2")

print("""Decision Tree Regressor \t {:.2f} \t\t {:.2f} \t\t{:.2f} \t\t{:.2f}""".format(

np.sqrt(mean\_squared\_error(y\_test, y\_pred\_dtr)),mean\_squared\_error(y\_test, y\_pred\_dtr),

mean\_absolute\_error(y\_test, y\_pred\_dtr), r2\_score(y\_test, y\_pred\_dtr)))

plt.scatter(y\_test, y\_pred\_dtr)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=2)

plt.xlabel("Predicted")

plt.ylabel("True")

plt.title("Decision Tree Regressor")

plt.show()

# In[41]:

#Random Forest Regressor

# In[42]:

from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor(n\_estimators=100)

rfr.fit(X\_train, y\_train)

y\_pred\_rfr = rfr.predict(X\_test)

print("Model\t\t\t\t RMSE \t\t MSE \t\t MAE \t\t R2")

print("""Random Forest Regressor \t {:.2f} \t\t {:.2f} \t\t{:.2f} \t\t{:.2f}""".format(

np.sqrt(mean\_squared\_error(y\_test, y\_pred\_rfr)),mean\_squared\_error(y\_test, y\_pred\_rfr),

mean\_absolute\_error(y\_test, y\_pred\_rfr), r2\_score(y\_test, y\_pred\_rfr)))

plt.scatter(y\_test, y\_pred\_rfr)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=2)

plt.xlabel("Predicted")

plt.ylabel("True")

plt.title("Decision Tree Regressor")

plt.show()

# In[43]:

# Multi Layer Perceptron

# In[44]:

from sklearn.neural\_network import MLPRegressor

mlp = MLPRegressor(hidden\_layer\_sizes=(100,50), max\_iter=1000)

mlp.fit(X\_train, y\_train)

y\_pred\_mlp = rfr.predict(X\_test)

print("Model\t\t\t\t RMSE \t\t MSE \t\t MAE \t\t R2")

print("""Multi Layer Perceptron \t\t {:.2f} \t\t {:.2f} \t\t{:.2f} \t\t{:.2f}""".format(

np.sqrt(mean\_squared\_error(y\_test, y\_pred\_mlp)),mean\_squared\_error(y\_test, y\_pred\_mlp),

mean\_absolute\_error(y\_test, y\_pred\_mlp), r2\_score(y\_test, y\_pred\_mlp)))

plt.scatter(y\_test, y\_pred\_mlp)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--', lw=2)

plt.xlabel("Predicted")

plt.ylabel("True")

plt.title("Decision Tree Regressor")

plt.show()

# In[45]:

#Comparision

# In[46]:

models = [lr, lasso, ridge, dtr, rfr, mlp]

names = ["Linear Regression", "Lasso Regression", "Ridge Regression",

"Decision Tree Regressor", "Random Forest Regressor", "Multi Layer Perceptron"]

rmses = []

for model in models:

rmses.append(np.sqrt(mean\_squared\_error(y\_test, model.predict(X\_test))))

x = np.arange(len(names))

width = 0.3

fig, ax = plt.subplots(figsize=(10,7))

rects = ax.bar(x, rmses, width)

ax.set\_ylabel('RMSE')

ax.set\_xlabel('Models')

ax.set\_title('RMSE with Different Algorithms')

ax.set\_xticks(x)

ax.set\_xticklabels(names, rotation=45)

autolabel(rects)

fig.tight\_layout()

plt.show()

# In[ ]:

Output:







