IIT BHU

1 March 2023

ARTIFICIAL INTELLIGENCE

Mid-Sem Evaluation

PRESENTED BY

Tushar Talesara
21074031

Vaibhav Khater
21074033

PROBLEM STATEMENT:

Prediction of Concrete Strengths Enabled by Missing Data Imputation and Interpretable Machine Learning

OVERVIEW :

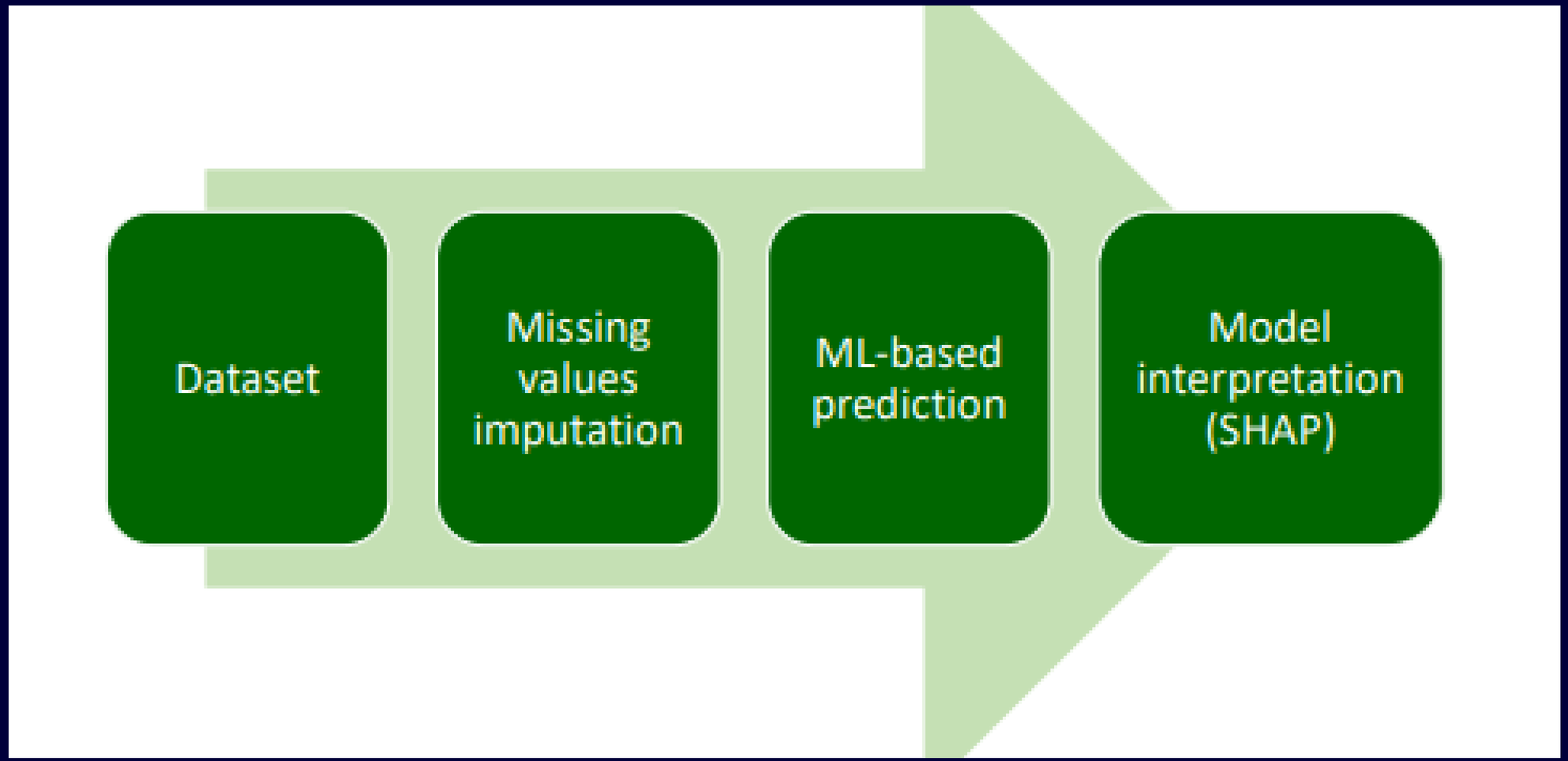
- Machine learning (ML)-based prediction of non-linear composition-strength relationship in concretes requires a large, complete, and consistent dataset. However, the availability of such datasets is limited as the datasets often suffer from incompleteness because of missing data corresponding to different input features, which makes the development of robust ML-based predictive models challenging. Besides, as the degree of complexity in ML models increases, the interpretation of the results becomes challenging.

- These interpretations of results are critical towards the development of efficient materials design strategies for enhanced materials performance. To address these challenges, this paper implements different data imputation approaches for enhanced dataset completeness. The imputed dataset is leveraged to predict the compressive and tensile strength of concrete using various hyperparameteroptimized ML

Why ML based data driven approach ?

- Conventional approaches have played significant roles in extensively linking the parameters such as cement dosage, aggregate fraction, and air void content with the concrete strength, evaluation of the combined effects of these features is still a challenging task. In addition, these conventional approaches ignore the influence of secondary factors such as the nature and dosage of chemical admixtures, aggregate size distribution, and fineness modulus of aggregates

- # METHODOLOGY :



DATASET DESCRIPTION :

- The dataset used in this study comprises 754 uniaxial compressive strength and splitting tensile strength data for manufactured sand concrete (MSC).
- The mixture of MSC consists of Grade 42.5 ordinary silicate cement, which is mixed with supplementary cementitious materials (SCMs), crushed stone, and manufactured sand. The SCMs include fly ash, slag, and silica fume.

- Attributes used in this datasets are :-

1. 28-day compressive strength of hardened cement paste that ranges from 35.5 to 63.4 MPa
2. 28-day tensile strength that ranges from 6.9 to 10.8 MPa, maximum size of the crushed stone varied from 12 mm to 120 mm
3. The fineness modulus of manufactured sand ranged from 2.2 to 3.55
4. The mass-based water-binder ratio ranged between 0.24- 1.00

5. A range of 0.3-1.43 was adopted for the water-cement ratio, sand ratio considered was 25-54%
6. The curing time varied from 3 days to 388 days
7. 28-day uniaxial compressive strength of concrete ranged from 10.1 - 96.3 MPa
8. 28-day splitting tensile strength ranged from 0.6 to 6.9 MPa

FEATURISATION METHODS USED IN THE PAPER:

- To build a robust ML model, the data provided should be free from missing values, duplicate entries, and outliers. However, in reality, the data are often embedded with uncertainties due to how the data were procured. One such prevalent case is the missing data. Training of the model on a dataset that has multiples missing values can significantly impact the quality of prediction.

- Toward this, the imputation of missing values can be adopted as an efficient alternative approach. In this study, the various data imputation methods are adopted to handle the missing values. These include removing the missing values, imputation using mean/median values, imputation using k-Nearest Neighbors (k-NN), and multivariate imputation by chained equations (MICE).

1. REMOVAL OF MISSING VALUES:

- In this method, the missing values are deleted from the dataset.
- LIMITATIONS :In general, this method is adopted only when the proportion of missing values is significantly small (<5%)

CODE FOR REMOVAL OF MISSING VALUES :

```
# Removing missing values
df_imputed=df.dropna(axis=0, how='any')
df_imputed.columns=df.columns
```

2. REPLACEMENT BY MEAN/MEDIAN:

- In this method, the missing value for each feature is replaced by the mean or median of the non-missing values of the respective features. Such a method is easy to implement and works well with small numerical datasets.
- LIMITATIONS : This method does not consider the correlation between features and does not account for the uncertainty in the imputations .

CODE FOR IMPUTATION USING MEAN :

```
# Imputation using mean
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
df_imputed = pd.DataFrame(imputer.fit_transform(df))
df_imputed.columns = df.columns
```

CODE FOR IMPUTATION USING MEDIAN :

```
# Imputation using median
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='median')
df_imputed = pd.DataFrame(imputer.fit_transform(df))
df_imputed.columns = df.columns
```

3. K-NEAREST NEIGHBOUR :

- The k-NN is an algorithm that uses the similarity in the features to predict the values of any new data points. In other words, the new point is assigned based on how closely it resembles the points in the training set. In k-NN based method [58], the missing value is imputed by taking the weighted average of the feature associated with the missing value from the k-closest sets.
- LIMITATIONS : it is sensitive to an outlier in the data

CODE FOR IMPUTATION USING KNN(5) :

```
# Imputation using kNN using 5 neighbours
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5)
df_imputed=pd.DataFrame(imputer.fit_transform(df))
df_imputed.columns=df.columns
```

CODE FOR IMPUTATION USING KNN(10) :

```
# Imputation using kNN using 10 neighbours
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=10)
df_imputed=pd.DataFrame(imputer.fit_transform(df))
df_imputed.columns=df.columns
```

4. MULTIVARIATE IMPUTATION BY CHAINED EQUATIONS (MICE) :

- MICE is an iterative approach where the missing value in a dataset is imputed based on errorminimization. MICE-based imputation approach adopts a series of estimations where each missing variable is imputed by regressing on the other variables. This is achieved by running through an iterative process.

- In the first iteration, the missing value is initially imputed by taking the mean of the observed values for that feature in the dataset. In the second iteration, the feature with the fewest missing values is selected, and the corresponding imputed mean values are removed from the dataset. In this iteration, the removed set is now considered as the test set, and the remaining dataset is considered as the training set. The first missing value in the test set is the dependent variable, whereas all the other values in the test set are considered independent variables.

- The independent variables from this test set are then imputed based on the regression model, which is developed based on the training set. In the third iteration, the previous steps are repeated for the remaining missing features, where the predicted feature from the previous iteration is also included as the independent variable. This entire process of iterating through the missing features is repeated until convergence is reached. Thus, the whole process is run multiple times to get multiple imputations

CODE FOR IMPUTATION USING MICE :

```
# Imputation using MICE
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import LinearRegression
imputer = IterativeImputer(estimator=LinearRegression(),missing_values=np.nan, max_iter=10, verbose=2, imputation_order='roman',random_state=0)
df_imputed=pd.DataFrame(imputer.fit_transform(df))
df_imputed.columns=df.columns
```

ALGORITHMS USED IN THE PAPER :

Extreme Gradient Boosted Decision Trees (XGBoost)

- After missing data imputation, the concrete strengths are predicted using ML techniques.
- XGBoost is an extreme gradient boosted and tree-based machine learning model. It is an iterative approach where the models are trained in succession to minimize the errors.

- The iteration continues till no further improvements can be made. It is also considered an ensemble technique in which many tree models are combined to obtain the final one. XGBoost contains a toolset for scalable end-to-end tree boosting systems, sparsity-based algorithms, and justified weighted sketch for efficient proposal calculation.

Mean square error (MSE) and Coefficient of determination (R 2)

- The predictive capability of the ML techniques is evaluated using both mean square error (MSE) and coefficient of determination (R 2).
- The MSE measures the average Euclidean distance between the predicted and true or measured values
- The coefficient of determination is used to quantify the variation of the predicted values from the observed values.

CODE :

- COEFFICIENT OF DETERMINATION :

```
from sklearn.metrics import r2_score
print("R2 accuracy",r2_score(y_test, y_pred))
```

- MEAN SQUARED ERROR :

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, y_pred)
print("Mean squared error: ", mse)
```

Hyperparameter tuning

- To increase the accuracy of the ML models and to avoid any potential overfitting, a fraction of the data points is kept fully hidden from the models and used as a “test set.” The test set is then used to evaluate the accuracy and performance of each model on these unseen data.
- To this end, a k-fold cross-validation (CV) technique is adopted , the dataset is split into k number of smaller sets, wherein at each fold, the model is trained on a fraction of data (training set) and tested on the remaining data.

- The nested two-level cross-validation (CV) is implemented to avoid any risk of over-fitting.
- First, in the outer CV (5-fold), 15% of the data is first randomly split from the total dataset to be used as the test set, which is kept completely hidden from the model training and validation.
- Next, in the inner CV (5-fold), the remaining 85% of the data is further split into training and validation sets at an 85:15 ratio.
- Thus, the nested CV technique discards any arbitrary choice of the test set and partially mitigatesissues arising from the limited number of data points.

QUANTUM ALGORITHMS USED BY US: :

1 . SUPPORT VECTOR MACHINE :

```
# SVM
from sklearn.svm import SVR
classifier = SVR()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

2. KNN(15) :

```
# Imputation using kNN using 15 neighbours
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=15)
df_imputed=pd.DataFrame(imputer.fit_transform(df))
df_imputed.columns=df.columns
```

FEATURISATION METHODS USED BY US :

1. STANDARD SCALAR :

```
# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
y_axis=df_imputed.iloc[:,[0]]
x_axis=df_imputed.iloc[:,[1,2,3,4,5,6,7,8,9,10,11,12]]
X_train, X_test, y_train, y_test = train_test_split(x_axis,y_axis, test_size=0.2, random_state=42)

# Featurisation
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

2. KNN(15) :

```
# Imputation using kNN using 15 neighbours
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=15)
df_imputed=pd.DataFrame(imputer.fit_transform(df))
df_imputed.columns=df.columns
```

OTHER ALGORITHMS USED BY US :

LINEAR REGRESSION :

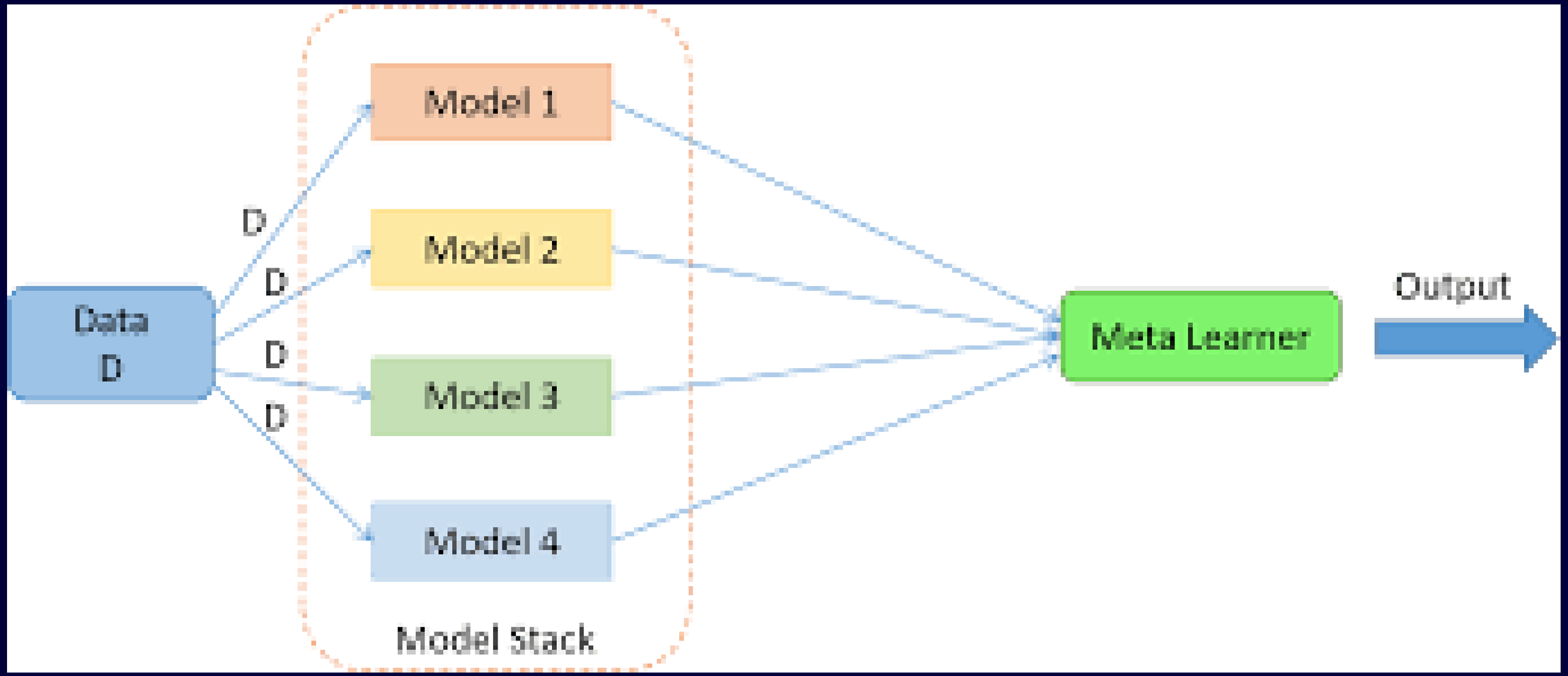
```
# Linear Regression
# Build the model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
# # Train the model
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

META MODEL :

- Meta-learning algorithms use learning algorithm metadata as input. They then make predictions and provide information about the performance of these learning algorithms as output.Meta learning is essentially learning how to learn .Meta learning algorithms learns from output of other learning algorithms which itself learn from data.

META MODEL

```
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
lin = LinearRegression()
rnd = RandomForestRegressor(n_estimators=100)
svm = SVR()
voting = VotingRegressor(estimators=[('linear_regression', lin), ('random_forest', rnd), ('support_vector_machine', svm)])
voting.fit(X_train, y_train)
```


Model 1
Model 2
Model 3
Model 4
Model Stack
Data
D
D
D
D
D
Meta Learner
Output

Thank You