friedman

March 21, 2023

[1]:
```
import pandas as pd
import numpy as np
from tensorflow.keras import layers
from tensorflow.keras import models
from sklearn.metrics import r2_score
```

[2]:
```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LinearRegression
from sklearn.neural_network import MLPRegressor
from xgboost import XGBRegressor
```

[3]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

[4]:
```
# to ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

0.0.1 With the help of Pandas, read the ".csv" file and performing some task

[5]:
```
data1 = pd.read_csv("friedman.csv")
```

[6]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1202 entries, 0 to 1201
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   @inputs Input1   1200 non-null   float64
 1   Input2           1200 non-null   float64
 2   Input3           1200 non-null   float64
 3   Input4           1200 non-null   float64
 4   Input5           1200 non-null   float64
 5   @outputs Output  1200 non-null   float64
```

```
dtypes: float64(6)
memory usage: 56.5 KB
```

[7]:
```
data1.rename(columns = {'@inputs Input1':'Input1','@outputs Output':'Output'},
↪inplace = True)
```

[8]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1202 entries, 0 to 1201
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Input1  1200 non-null   float64
 1   Input2  1200 non-null   float64
 2   Input3  1200 non-null   float64
 3   Input4  1200 non-null   float64
 4   Input5  1200 non-null   float64
 5   Output  1200 non-null   float64
dtypes: float64(6)
memory usage: 56.5 KB
```

[9]:
```
data1.head(4)
```

[9]:

	Input1	Input2	Input3	Input4	Input5	Output
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN
2	0.696482	0.358437	0.425834	0.330314	0.222491	11.094962
3	0.590390	0.430675	0.869042	0.070912	0.634303	13.229209

[10]:
```
data1.tail(4)
```

[10]:

	Input1	Input2	Input3	Input4	Input5	Output
1198	0.348151	0.406174	0.243864	0.201591	0.040682	7.071847
1199	0.839787	0.759799	0.193053	0.187603	0.658195	14.336152
1200	0.017182	0.959536	0.815701	0.213163	0.681054	8.318943
1201	0.347079	0.870634	0.706439	0.147060	0.489136	13.031322

[11]:
```
data1.isnull().any()
```

[11]:
```
Input1    True
Input2    True
Input3    True
Input4    True
Input5    True
Output    True
dtype: bool
```

[12]:
```python
data1.isnull().sum()
```

[12]:
```
Input1    2
Input2    2
Input3    2
Input4    2
Input5    2
Output    2
dtype: int64
```

[13]:
```python
null_cols = data1.isnull().sum()
```

[14]:
```python
# Loop through each input column and impute with mean
for col in data1.columns[:-1]:
    mean_val = data1[col].mean()
    data1[col].fillna(mean_val, inplace=True)
```

[15]:
```python
data1.isnull().sum()
```

[15]:
```
Input1    0
Input2    0
Input3    0
Input4    0
Input5    0
Output    2
dtype: int64
```

[16]:
```python
# Impute null values in output column with mean
mean_val1 = data1['Output'].mean()
data1['Output'].fillna(mean_val1, inplace=True)
```

[17]:
```python
data1.isnull().sum()
```

[17]:
```
Input1    0
Input2    0
Input3    0
Input4    0
Input5    0
Output    0
dtype: int64
```

[18]:
```python
import matplotlib.pyplot as plt
```

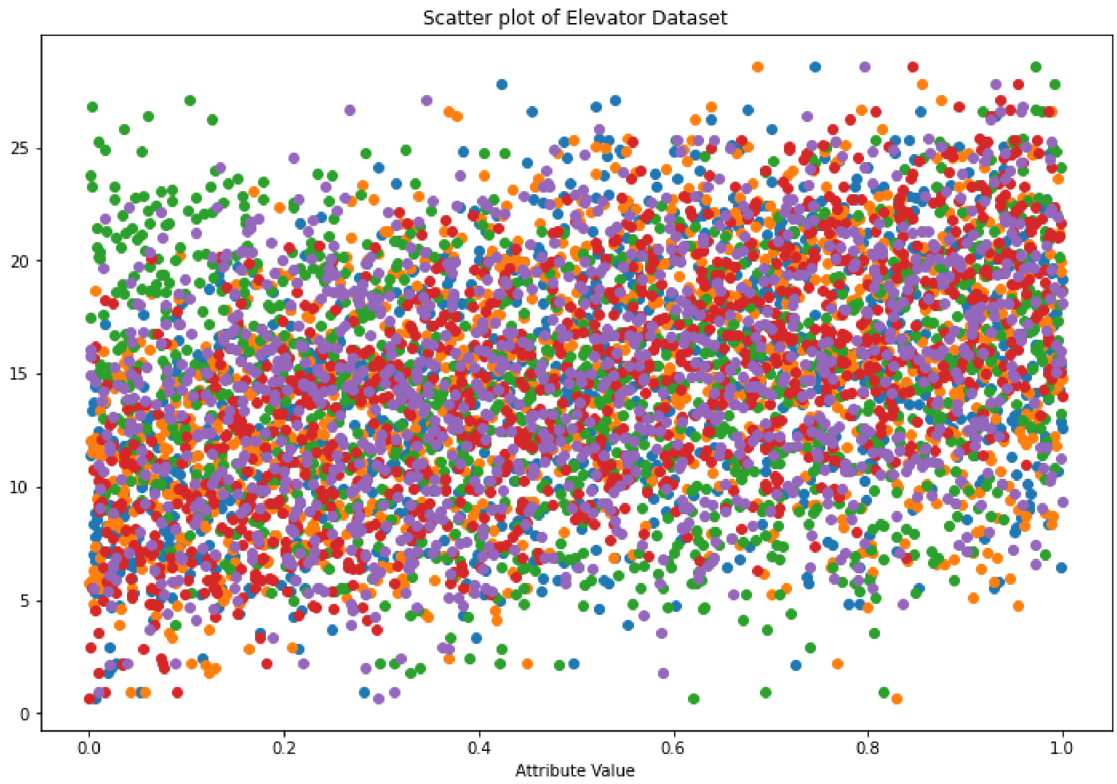
[19]:
```python
fig, ax = plt.subplots(figsize=(12,8))

ax.scatter(data1["Input1"], data1["Output"])
ax.scatter(data1["Input2"], data1["Output"])
ax.scatter(data1["Input3"], data1["Output"])
```

```
ax.scatter(data1["Input4"], data1["Output"])
ax.scatter(data1["Input5"], data1["Output"])


ax.set_xlabel("Attribute Value")

ax.set_title("Scatter plot of Elevator Dataset")
plt.show()
```



[20]:
```
X = data1.drop('Output', axis=1).values
y = data1['Output'].values
```

0.0.2 Create a linear regression model

[21]:
```
linear_model = LinearRegression()
```

[22]:
```
# Define the cross-validation method
cv = KFold(n_splits=5, shuffle=True, random_state=42)
```

[23]:
```
# Evaluate the model using 5-fold cross-validation
linear_scores = cross_val_score(linear_model, X, y, cv=cv,␣
↪scoring='neg_mean_squared_error')
```

[24]:
```
# Calculate the evaluation metrics from the scores
rmse = np.sqrt(-linear_scores.mean())
mse = -linear_scores.mean()
```

[25]:
```
mae = -cross_val_score(linear_model, X, y, cv=cv,␣
↪scoring='neg_mean_absolute_error').mean()
r2 = cross_val_score(linear_model, X, y, cv=cv, scoring='r2').mean()
```

[26]:
```
#the Linear Regression model
print('Linear Regression Model:')
print('RMSE:', rmse)
print('MSE:', mse)
print('MAE:', mae)
print('R-squared:', r2)
```

```
Linear Regression Model:
RMSE: 2.6962932503590205
MSE: 7.269997291931611
MAE: 2.0845673089752363
R-squared: 0.7281145447623009
```

0.0.3 Create an Artificial Neural Network model

[27]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[28]:
```
# Scale the features using standard scaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[29]:
```
# Create an Artificial Neural Network model
model = Sequential()
model.add(Dense(units=16, activation='relu', input_dim=X_train.shape[1]))
model.add(Dropout(rate=0.2))
model.add(Dense(units=8, activation='relu'))
model.add(Dropout(rate=0.2))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
```

[30]:
```
# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=64, validation_split=0.2)
```

```
Epoch 1/50
12/12 [==============================] - 1s 17ms/step - loss: 252.3782 -
val_loss: 259.0193
Epoch 2/50
```

```
12/12 [==============================] - 0s 6ms/step - loss: 247.1452 -
val_loss: 254.3651
Epoch 3/50
12/12 [==============================] - 0s 5ms/step - loss: 243.3380 -
val_loss: 250.4047
Epoch 4/50
12/12 [==============================] - 0s 5ms/step - loss: 238.9238 -
val_loss: 246.8733
Epoch 5/50
12/12 [==============================] - 0s 5ms/step - loss: 236.0591 -
val_loss: 243.5087
Epoch 6/50
12/12 [==============================] - 0s 5ms/step - loss: 232.5696 -
val_loss: 240.1729
Epoch 7/50
12/12 [==============================] - 0s 5ms/step - loss: 228.8369 -
val_loss: 236.8169
Epoch 8/50
12/12 [==============================] - 0s 7ms/step - loss: 226.4194 -
val_loss: 233.3858
Epoch 9/50
12/12 [==============================] - 0s 5ms/step - loss: 222.1881 -
val_loss: 229.5893
Epoch 10/50
12/12 [==============================] - 0s 5ms/step - loss: 218.6411 -
val_loss: 225.3600
Epoch 11/50
12/12 [==============================] - 0s 5ms/step - loss: 213.2106 -
val_loss: 220.6633
Epoch 12/50
12/12 [==============================] - 0s 4ms/step - loss: 209.2464 -
val_loss: 215.5111
Epoch 13/50
12/12 [==============================] - 0s 4ms/step - loss: 204.3389 -
val_loss: 209.9335
Epoch 14/50
12/12 [==============================] - 0s 6ms/step - loss: 198.9249 -
val_loss: 203.8828
Epoch 15/50
12/12 [==============================] - 0s 5ms/step - loss: 192.0374 -
val_loss: 197.2058
Epoch 16/50
12/12 [==============================] - 0s 4ms/step - loss: 185.2454 -
val_loss: 189.8134
Epoch 17/50
12/12 [==============================] - 0s 4ms/step - loss: 179.9406 -
val_loss: 181.8738
Epoch 18/50
```

```
12/12 [==============================] - 0s 4ms/step - loss: 172.0155 -
val_loss: 173.4478
Epoch 19/50
12/12 [==============================] - 0s 5ms/step - loss: 165.0552 -
val_loss: 164.5048
Epoch 20/50
12/12 [==============================] - 0s 4ms/step - loss: 153.6075 -
val_loss: 154.9550
Epoch 21/50
12/12 [==============================] - 0s 5ms/step - loss: 145.7170 -
val_loss: 144.9520
Epoch 22/50
12/12 [==============================] - 0s 6ms/step - loss: 138.1663 -
val_loss: 134.5430
Epoch 23/50
12/12 [==============================] - 0s 6ms/step - loss: 126.6693 -
val_loss: 123.8385
Epoch 24/50
12/12 [==============================] - 0s 5ms/step - loss: 118.1079 -
val_loss: 113.0589
Epoch 25/50
12/12 [==============================] - 0s 5ms/step - loss: 108.6565 -
val_loss: 102.2425
Epoch 26/50
12/12 [==============================] - 0s 5ms/step - loss: 97.0588 - val_loss:
91.5823
Epoch 27/50
12/12 [==============================] - 0s 5ms/step - loss: 90.7468 - val_loss:
81.2450
Epoch 28/50
12/12 [==============================] - 0s 5ms/step - loss: 80.4169 - val_loss:
71.7176
Epoch 29/50
12/12 [==============================] - 0s 5ms/step - loss: 73.8794 - val_loss:
62.7222
Epoch 30/50
12/12 [==============================] - 0s 5ms/step - loss: 69.9019 - val_loss:
54.5315
Epoch 31/50
12/12 [==============================] - 0s 5ms/step - loss: 60.5964 - val_loss:
47.2249
Epoch 32/50
12/12 [==============================] - 0s 5ms/step - loss: 53.9876 - val_loss:
40.9681
Epoch 33/50
12/12 [==============================] - 0s 5ms/step - loss: 52.7356 - val_loss:
35.6330
Epoch 34/50
```

```
12/12 [==============================] - 0s 6ms/step - loss: 54.1612 - val_loss:
31.3898
Epoch 35/50
12/12 [==============================] - 0s 5ms/step - loss: 48.9563 - val_loss:
27.9777
Epoch 36/50
12/12 [==============================] - 0s 5ms/step - loss: 48.6877 - val_loss:
25.1049
Epoch 37/50
12/12 [==============================] - 0s 5ms/step - loss: 44.6709 - val_loss:
23.0617
Epoch 38/50
12/12 [==============================] - 0s 5ms/step - loss: 44.5108 - val_loss:
21.4052
Epoch 39/50
12/12 [==============================] - 0s 5ms/step - loss: 44.2013 - val_loss:
19.7517
Epoch 40/50
12/12 [==============================] - 0s 5ms/step - loss: 41.0403 - val_loss:
18.5326
Epoch 41/50
12/12 [==============================] - 0s 4ms/step - loss: 43.9875 - val_loss:
17.8060
Epoch 42/50
12/12 [==============================] - 0s 5ms/step - loss: 43.9410 - val_loss:
16.9510
Epoch 43/50
12/12 [==============================] - 0s 5ms/step - loss: 45.4846 - val_loss:
16.3857
Epoch 44/50
12/12 [==============================] - 0s 4ms/step - loss: 40.5152 - val_loss:
16.1031
Epoch 45/50
12/12 [==============================] - 0s 4ms/step - loss: 38.0606 - val_loss:
15.6306
Epoch 46/50
12/12 [==============================] - 0s 4ms/step - loss: 42.5571 - val_loss:
15.1468
Epoch 47/50
12/12 [==============================] - 0s 5ms/step - loss: 41.5191 - val_loss:
14.8363
Epoch 48/50
12/12 [==============================] - 0s 4ms/step - loss: 42.1334 - val_loss:
14.5936
Epoch 49/50
12/12 [==============================] - 0s 4ms/step - loss: 42.1698 - val_loss:
14.7882
Epoch 50/50
```

```
12/12 [==============================] - 0s 5ms/step - loss: 38.6451 - val_loss:
14.9994
```

[30]:
```
<keras.callbacks.History at 0x19a8c1f6260>
```

[31]:
```python
# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
8/8 [==============================] - 0s 1ms/step
```

[32]:
```python
# Calculate the evaluation metrics
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mse = mean_squared_error(y_test, y_pred)
```

[33]:
```python
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

[34]:
```python
# Print the evaluation metrics for the Artificial Neural Network model
print('Artificial Neural Network Model:')
print('RMSE:', rmse)
print('MSE:', mse)
print('MAE:', mae)
print('R-squared:', r2)
```

```
Artificial Neural Network Model:
RMSE: 3.678429963996212
MSE: 13.530847000025176
MAE: 2.9567028927979604
R-squared: 0.4610289809575676
```

0.0.4 Create an XGBoost Regression model

[35]:
```python
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)
```

[36]:
```python
# Evaluate the model using 5-fold cross-validation
xgb_scores = cross_val_score(xgb_model, X, y, cv=cv,␣
 ↪scoring='neg_mean_squared_error')
```

[37]:
```python
# Calculate the evaluation metrics from the scores
xgb_rmse = np.sqrt(-xgb_scores.mean())
xgb_mse = -xgb_scores.mean()
```

[38]:
```python
xgb_mae = -cross_val_score(xgb_model, X, y, cv=cv,␣
 ↪scoring='neg_mean_absolute_error').mean()
xgb_r2 = cross_val_score(xgb_model, X, y, cv=cv, scoring='r2').mean()
```

[39]:
```python
# Print the evaluation metrics for the XGBoost Regression model
print('XGBoost Regression Model:')
print('RMSE:', xgb_rmse)
```

```python
print('MSE:', xgb_mse)
print('MAE:', xgb_mae)
print('R-squared:', xgb_r2)
```

```
XGBoost Regression Model:
RMSE: 1.6957990140851462
MSE: 2.875734296172154
MAE: 1.318473382554611
R-squared: 0.8921737087350579
```

[40]:
```python
import seaborn as sns
```

[41]:
```python
sns.set_style("whitegrid")

# Create the lmplot with height=8 and aspect=1.5
sns.lmplot(x='Input1', y='Output', data=data1, scatter_kws={'color': 'red'},␣
↪line_kws={'color': 'blue'}, height=8, aspect=1.5)

# Show the plot
plt.show()
```