Concrete Compressive Strength Analysis with Keras

July 12, 2024

Concrete Compressive Strength Analysis with Keras

[372]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from keras.models import Sequential
from keras.layers import Dense, Input
from keras.optimizers import Adam
from keras.regularizers import l2
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score
```

0.1 Download and Clean Dataset

[360]:
```
data = pd.read_csv('https://cocl.us/concrete_data')
data.head()
```

[360]:
```
   Cement  Blast Furnace Slag  Fly Ash  Water  Superplasticizer  \
0   540.0                 0.0      0.0  162.0               2.5
1   540.0                 0.0      0.0  162.0               2.5
2   332.5               142.5      0.0  228.0               0.0
3   332.5               142.5      0.0  228.0               0.0
4   198.6               132.4      0.0  192.0               0.0

   Coarse Aggregate  Fine Aggregate  Age  Strength
0            1040.0           676.0   28     79.99
1            1055.0           676.0   28     61.89
2             932.0           594.0  270     40.27
3             932.0           594.0  365     41.05
4             978.4           825.5  360     44.30
```

[361]:
```
data.shape
```

[361]:
```
(1030, 9)
```

[362]:
```
data.describe()
```

[362]:

	Cement	Blast Furnace Slag	Fly Ash	Water
count	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.895825	54.188350	181.567282
std	104.506364	86.279342	63.997004	21.354219
min	102.000000	0.000000	0.000000	121.800000
25%	192.375000	0.000000	0.000000	164.900000
50%	272.900000	22.000000	0.000000	185.000000
75%	350.000000	142.950000	118.300000	192.000000
max	540.000000	359.400000	200.100000	247.000000

	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
count	1030.000000	1030.000000	1030.000000	1030.000000
mean	6.204660	972.918932	773.580485	45.662136
std	5.973841	77.753954	80.175980	63.169912
min	0.000000	801.000000	594.000000	1.000000
25%	0.000000	932.000000	730.950000	7.000000
50%	6.400000	968.000000	779.500000	28.000000
75%	10.200000	1029.400000	824.000000	56.000000
max	32.200000	1145.000000	992.600000	365.000000

	Strength
count	1030.000000
mean	35.817961
std	16.705742
min	2.330000
25%	23.710000
50%	34.445000
75%	46.135000
max	82.600000

Checking null items

[363]:
```
data.isnull().sum()
```

[363]:
```
Cement                0
Blast Furnace Slag    0
Fly Ash               0
Water                 0
Superplasticizer      0
Coarse Aggregate      0
Fine Aggregate        0
Age                   0
Strength              0
dtype: int64
```

0.2 Calculating and Using VIF to Drop Variables

[ ]:
```
"""
# Ensure all columns are numeric before calculating VIF
numeric_data = data.select_dtypes(include=[float, int])

# Calculate VIF for each feature
def calculate_vif(df):
    vif = pd.DataFrame()
    vif["Features"] = df.columns
    vif["VIF"] = [variance_inflation_factor(df.values, i) for i in range(df.
 ↪shape[1])]
    return vif

# Calculate VIF and drop highly correlated features
vif = calculate_vif(numeric_data.drop('Strength', axis=1))
print(vif)

# Dropping features with VIF greater than 10
high_vif_features = vif[vif["VIF"] > 10]["Features"].tolist()
data_dropped = data.drop(columns=high_vif_features)

data_dropped.head()

"""
```

[373]:
```
# Define features and target
X = data.drop('Strength', axis=1)
y = data['Strength']
X.head()
```

[373]:
```
   Cement  Blast Furnace Slag  Fly Ash  Water  Superplasticizer  \
0   540.0                 0.0      0.0  162.0               2.5
1   540.0                 0.0      0.0  162.0               2.5
2   332.5               142.5      0.0  228.0               0.0
3   332.5               142.5      0.0  228.0               0.0
4   198.6               132.4      0.0  192.0               0.0

   Coarse Aggregate  Fine Aggregate  Age
0            1040.0           676.0   28
1            1055.0           676.0   28
2             932.0           594.0  270
3             932.0           594.0  365
4             978.4           825.5  360
```

[374]:
```
y.head()
```

[374]:
```
0    79.99
1    61.89
2    40.27
3    41.05
4    44.30
Name: Strength, dtype: float64
```

0.3 Normalizing the data

[375]:
```python
scaler = StandardScaler()
X_norm = scaler.fit_transform(X)
```

[376]:
```python
X_norm
```

[376]:
```
array([[ 2.47791487, -0.85688789, -0.84714393, …,  0.86315424,
        -1.21767004, -0.27973311],
       [ 2.47791487, -0.85688789, -0.84714393, …,  1.05616419,
        -1.21767004, -0.27973311],
       [ 0.49142531,  0.79552649, -0.84714393, …, -0.52651741,
        -2.24091709,  3.55306569],
       …,
       [-1.27008832,  0.75957923,  0.85063487, …, -1.03606368,
         0.0801067 , -0.27973311],
       [-1.16860982,  1.30806485, -0.84714393, …,  0.21464081,
         0.19116644, -0.27973311],
       [-0.19403325,  0.30849909,  0.3769452 , …, -1.39506219,
        -0.15074782, -0.27973311]])
```

0.4 Split data into train and test

[377]:
```python
X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.3,␣
↪random_state=30)
```

0.5 Building the Neural Network

[378]:
```python
# Define the model
model = Sequential()

# Add the input layer with specified shape
model.add(Input(shape=(X_train.shape[1],)))

# Add additional layers as needed
model.add(Dense(640, activation='relu', kernel_regularizer=l2(0.00001)))
model.add(Dense(320, activation='relu', kernel_regularizer=l2(0.00001)))
model.add(Dense(10, activation='relu', kernel_regularizer=l2(0.00001)))
model.add(Dense(1, activation='linear'))
```

```
# Compile the model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Print model summary
model.summary()
```

```
Model: "sequential_37"


 Layer (type)                        Output Shape                          ␣
↪Param #

 dense_178 (Dense)                   (None, 640)                            ␣
↪5,760

 dense_179 (Dense)                   (None, 320)                          ␣
↪205,120

 dense_180 (Dense)                   (None, 10)                             ␣
↪3,210

 dense_181 (Dense)                   (None, 1)                                ␣
↪ 11



 Total params: 214,101 (836.33 KB)


 Trainable params: 214,101 (836.33 KB)


 Non-trainable params: 0 (0.00 B)
```

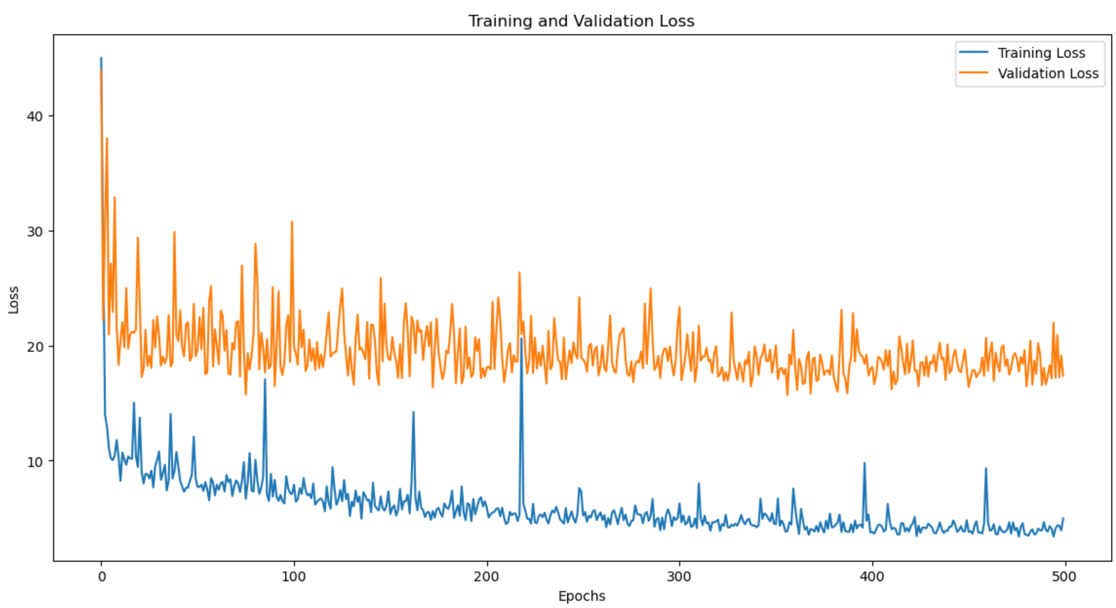
0.6 Training the Neural Network

[379]:
```
# Ensure all data types are correct
X_train = X_train.astype(float)
X_test = X_test.astype(float)
y_train = y_train.astype(float)
y_test = y_test.astype(float)

# Train the model
history = model.fit(X_train, y_train, epochs=500, verbose=0,␣
 ↪validation_data=(X_test, y_test), batch_size=8)
```

0.7 Evaluating the Model

[296]:
```
# Evaluate the model
loss, mae = model.evaluate(X_test, y_test)
print(f"Mean Absolute Error: {mae}")

plt.figure(figsize=(14, 7))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```

```
10/10              0s 3ms/step - loss:
20.4962 - mae: 2.9416
Mean Absolute Error: 2.799144983291626
```



[297]:
```
# Predict the Concrete Compressive Strength for the test dataset
y_pred = model.predict(X_test)


# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# R-squared
r2 = r2_score(y_test, y_pred)
```

```
10/10              0s 2ms/step
```

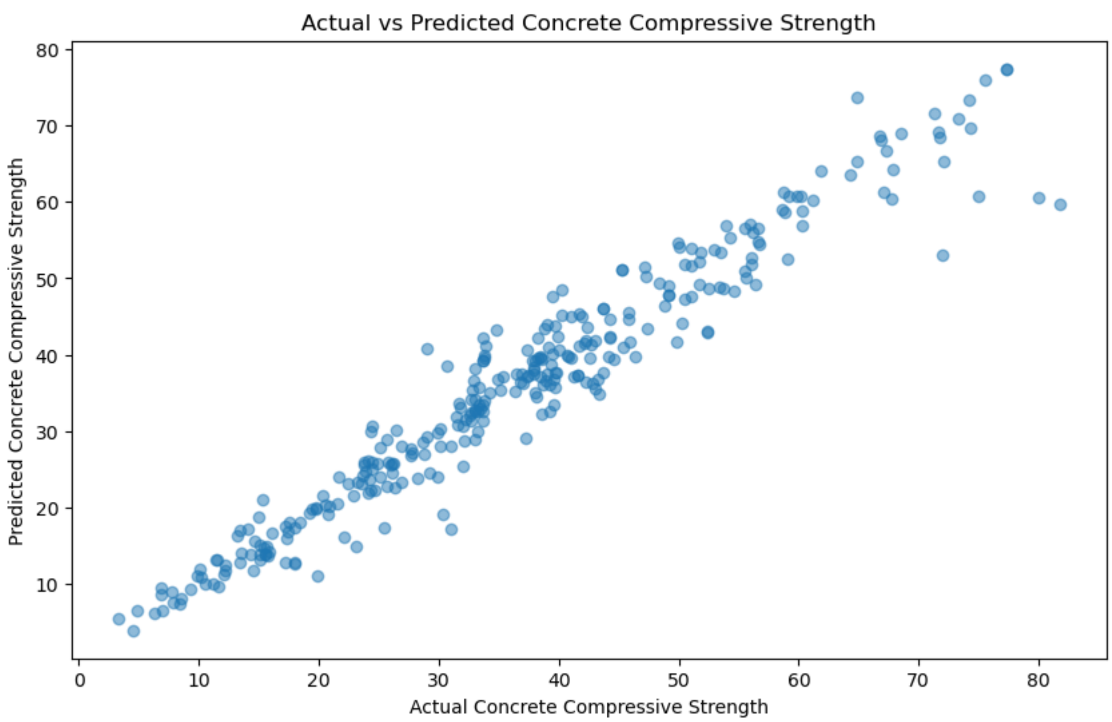
[298]:
```
r2
```

[298]:
```
0.9388676069557774
```

[299]:
```
mse
```

[299]:
```
17.34805599959136
```

[300]:
```python
# Plotting the Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel('Actual Concrete Compressive Strength')
plt.ylabel('Predicted Concrete Compressive Strength')
plt.title('Actual vs Predicted Concrete Compressive Strength')
plt.show()
```



[ ]: