Testing

November 6, 2024

1 COMP1801 - Machine Learning Coursework Solution

Let's start by importing the essential Python libraries for data analysis and machine learning.

[ ]:
```
# Import libraries
try:
    # Importing general libraries
    import glob
    import pandas as pd

    # Importing libraries for model building
    from sklearn.preprocessing import LabelEncoder
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import RandomForestRegressor
    from sklearn.metrics import root_mean_squared_error, r2_score

except Exception as e:
    print(f"Error : {e}")
```

[209]:
```
# Find the CSV file in the Datasets directory
data_path = '../Datasets/*.csv'
file_list = glob.glob(data_path)

for file in file_list:
    print(f"Found file: {file}")

# Ensure there is exactly one file
if len(file_list) == 1:
    # Load the dataset
    df = pd.read_csv(file_list[0])
    print(f"Loaded dataset: {file_list[0]}")
else:
    raise FileNotFoundError("No CSV file found or multiple CSV files found in␣
 ↪the Datasets directory.")
```

```
Found file: ../Datasets/Dataset.csv
Loaded dataset: ../Datasets/Dataset.csv
```

[210]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Lifespan        1000 non-null   float64
 1   partType        1000 non-null   object 
 2   microstructure  1000 non-null   object 
 3   coolingRate     1000 non-null   int64  
 4   quenchTime      1000 non-null   float64
 5   forgeTime       1000 non-null   float64
 6   HeatTreatTime   1000 non-null   float64
 7   Nickel%         1000 non-null   float64
 8   Iron%           1000 non-null   float64
 9   Cobalt%         1000 non-null   float64
 10  Chromium%       1000 non-null   float64
 11  smallDefects    1000 non-null   int64  
 12  largeDefects    1000 non-null   int64  
 13  sliverDefects   1000 non-null   int64  
 14  seedLocation    1000 non-null   object 
 15  castType        1000 non-null   object 
dtypes: float64(8), int64(4), object(4)
memory usage: 125.1+ KB
```

[211]:
```
# Check for missing values
df.isnull().sum()
```

[211]:
```
Lifespan          0
partType          0
microstructure    0
coolingRate       0
quenchTime        0
forgeTime         0
HeatTreatTime     0
Nickel%           0
Iron%             0
Cobalt%           0
Chromium%         0
smallDefects      0
largeDefects      0
sliverDefects     0
seedLocation      0
castType          0
dtype: int64
```

[212]:
```
df.head()
```

[212]:

	Lifespan	partType	microstructure	coolingRate	quenchTime	forgeTime	HeatTreatTime	Nickel%	Iron%	Cobalt%	Chromium%	smallDefects	largeDefects	sliverDefects	seedLocation	castType
0	1469.17	Nozzle	equiGrain	13	3.84	6.47	46.87	65.73	16.52	16.82	0.93	10	0	0	Bottom	Die
1	1793.64	Block	singleGrain	19	2.62	3.48	44.70	54.22	35.38	6.14	4.26	19	0	0	Bottom	Investment
2	700.60	Blade	equiGrain	28	0.76	1.34	9.54	51.83	35.95	8.81	3.41	35	3	0	Bottom	Investment
3	1082.10	Nozzle	colGrain	9	2.01	2.19	20.29	57.03	23.33	16.86	2.78	0	1	0	Top	Continuous
4	1838.83	Blade	colGrain	16	4.13	3.87	16.13	59.62	27.37	11.45	1.56	10	0	0	Top	Die

[213]:
```
df.describe()
```

[213]:

	Lifespan	coolingRate	quenchTime	forgeTime	HeatTreatTime	Nickel%	Iron%	Cobalt%	Chromium%	smallDefects	largeDefects	sliverDefects
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	1298.556320	17.639000	2.764230	5.464600	30.194510	60.243080	24.553580	12.434690	2.768650	17.311000	0.550000	0.292000
std	340.071434	7.491783	1.316979	2.604513	16.889415	5.790475	7.371737	4.333197	1.326496	12.268365	1.163982	1.199239
min	417.990000	5.000000	0.500000	1.030000	1.030000	50.020000	6.660000	5.020000	0.510000	0.000000		
25%	1047.257500	11.000000	1.640000	3.170000	16.185000	55.287500	19.387500	8.597500	1.590000	7.000000		
50%	1266.040000	18.000000	2.755000	5.475000	29.365000	60.615000	24.690000	12.585000	2.865000	18.000000		
75%	1563.050000	24.000000	3.970000	7.740000	44.955000	65.220000	29.882500	16.080000	3.922500	26.000000		
max	2134.530000	30.000000	4.990000	10.000000	59.910000	69.950000	43.650000	19.990000	4.990000	61.000000		

```
min         0.000000        0.000000
25%         0.000000        0.000000
50%         0.000000        0.000000
75%         0.000000        0.000000
max         4.000000        8.000000
```

[214]:
```
# Using nunique()
num_parts = df['partType'].nunique()
print(f"Number of unique parts types: {num_parts}")

# Or using value_counts() to see the distribution
parts_distribution = df['partType'].value_counts()
print("\nDistribution of parts types:")
print(parts_distribution)
```

```
Number of unique parts types: 4

Distribution of parts types:
partType
Valve     265
Block     253
Nozzle    245
Blade     237
Name: count, dtype: int64
```

[215]:
```
categorical_cols_unfied = ['partType', 'microstructure', 'seedLocation',␣
 ↪'castType']

# Create a DataFrame to display unique values and their counts
unique_values_df = pd.DataFrame({
    'Column': categorical_cols_unfied,
    'Unique Values': [df[col].unique().tolist() for col in␣
 ↪categorical_cols_unfied],
    'Count of Unique Values': [df[col].nunique() for col in␣
 ↪categorical_cols_unfied]
})

print(unique_values_df)
```

```
           Column                        Unique Values  Count of Unique Values
0        partType       [Nozzle, Block, Blade, Valve]                       4
1  microstructure  [equiGrain, singleGrain, colGrain]                       3
2    seedLocation                        [Bottom, Top]                       2
3        castType       [Die, Investment, Continuous]                       3
```

[216]:
```
# Creating a copy of the dataframe to ensure we maintain the original intact
df_onehot_encoded = df.copy()
```

```
# Apply one-hot encoding to the categorical columns
df_onehot_encoded = pd.get_dummies(df_onehot_encoded,␣
 ↪columns=categorical_cols_unfied, drop_first=False)

# Display the first few rows to verify
display(df_onehot_encoded.head())
```

```
   Lifespan  coolingRate  quenchTime  forgeTime  HeatTreatTime  Nickel%  \
0   1469.17           13        3.84       6.47          46.87    65.73
1   1793.64           19        2.62       3.48          44.70    54.22
2    700.60           28        0.76       1.34           9.54    51.83
3   1082.10            9        2.01       2.19          20.29    57.03
4   1838.83           16        4.13       3.87          16.13    59.62

   Iron%  Cobalt%  Chromium%  smallDefects  …  partType_Nozzle  \
0  16.52    16.82       0.93            10  …             True
1  35.38     6.14       4.26            19  …            False
2  35.95     8.81       3.41            35  …            False
3  23.33    16.86       2.78             0  …             True
4  27.37    11.45       1.56            10  …            False

   partType_Valve  microstructure_colGrain  microstructure_equiGrain  \
0           False                    False                      True
1           False                    False                     False
2           False                    False                      True
3           False                     True                     False
4           False                     True                     False

   microstructure_singleGrain  seedLocation_Bottom  seedLocation_Top  \
0                       False                 True             False
1                        True                 True             False
2                       False                 True             False
3                       False                False              True
4                       False                False              True

   castType_Continuous  castType_Die  castType_Investment
0                False          True                False
1                False         False                 True
2                False         False                 True
3                 True         False                False
4                False          True                False

[5 rows x 24 columns]
```

[217]:
```
df_onehot_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
```

```
Data columns (total 24 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Lifespan                     1000 non-null   float64
 1   coolingRate                  1000 non-null   int64
 2   quenchTime                   1000 non-null   float64
 3   forgeTime                    1000 non-null   float64
 4   HeatTreatTime                1000 non-null   float64
 5   Nickel%                      1000 non-null   float64
 6   Iron%                        1000 non-null   float64
 7   Cobalt%                      1000 non-null   float64
 8   Chromium%                    1000 non-null   float64
 9   smallDefects                 1000 non-null   int64
 10  largeDefects                 1000 non-null   int64
 11  sliverDefects                1000 non-null   int64
 12  partType_Blade               1000 non-null   bool
 13  partType_Block               1000 non-null   bool
 14  partType_Nozzle              1000 non-null   bool
 15  partType_Valve               1000 non-null   bool
 16  microstructure_colGrain      1000 non-null   bool
 17  microstructure_equiGrain     1000 non-null   bool
 18  microstructure_singleGrain   1000 non-null   bool
 19  seedLocation_Bottom          1000 non-null   bool
 20  seedLocation_Top             1000 non-null   bool
 21  castType_Continuous          1000 non-null   bool
 22  castType_Die                 1000 non-null   bool
 23  castType_Investment          1000 non-null   bool
dtypes: bool(12), float64(8), int64(4)
memory usage: 105.6 KB
```

[218]:
```python
# Define the target variable and feature set
X = df_onehot_encoded.drop(columns=['Lifespan'])  # Features
y = df_onehot_encoded['Lifespan']  # Target

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Display the shapes of the training and testing sets to verify
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (800, 23)
X_test shape: (200, 23)
y_train shape: (800,)
y_test shape: (200,)
```

[219]:
```
# Initialize the Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training data
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_model.predict(X_test)

# Evaluate the model using the new root_mean_squared_error function
rmse = root_mean_squared_error(y_test, y_pred)  # New recommended function
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R\u00b2 Score: {r2:.2f}")
```

```
Root Mean Squared Error (RMSE): 85.15
R² Score: 0.93
```

[220]:
```
# Creating a copy of the dataframe to ensure we maintain the original intact
df_label_encoded = df.copy()

# Apply Label Encoding to each categorical column
label_encoders = {}
for col in categorical_cols_unfied:
    le = LabelEncoder()
    df_label_encoded[col] = le.fit_transform(df_label_encoded[col])
    label_encoders[col] = le  # Store the encoder for inverse transformation if␣
 ↪needed later

# Display the first few rows to verify
display(df_label_encoded.head())
```

```
   Lifespan  partType  microstructure  coolingRate  quenchTime  forgeTime  \
0   1469.17         2               1           13        3.84       6.47
1   1793.64         1               2           19        2.62       3.48
2    700.60         0               1           28        0.76       1.34
3   1082.10         2               0            9        2.01       2.19
4   1838.83         0               0           16        4.13       3.87

   HeatTreatTime  Nickel%  Iron%  Cobalt%  Chromium%  smallDefects  \
0          46.87    65.73  16.52    16.82       0.93            10
1          44.70    54.22  35.38     6.14       4.26            19
2           9.54    51.83  35.95     8.81       3.41            35
3          20.29    57.03  23.33    16.86       2.78             0
4          16.13    59.62  27.37    11.45       1.56            10

   largeDefects  sliverDefects  seedLocation  castType
```

0	0	0	0	1
1	0	0	0	2
2	3	0	0	2
3	1	0	1	0
4	0	0	1	1

[221]:
```
# Define the target variable and feature set
X = df_label_encoded.drop(columns=['Lifespan'])  # Features
y = df_label_encoded['Lifespan']  # Target

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Display the shapes of the training and testing sets to verify
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (800, 15)
X_test shape: (200, 15)
y_train shape: (800,)
y_test shape: (200,)
```

[222]:
```
# Initialize the Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training data
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_model.predict(X_test)

# Evaluate the model using the new root_mean_squared_error function
rmse = root_mean_squared_error(y_test, y_pred)  # New recommended function
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R\u00b2 Score: {r2:.2f}")
```

```
Root Mean Squared Error (RMSE): 90.95
R² Score: 0.92
```