Build a Feature Engineering Regression Pipeline

By Josh Houlding

1. Formulate a prediction question that you want to answer by applying regression modeling. Examples: Prediction Question: How accurately can I predict the price of a house given the values of all the variables?
2. Search and locate a dataset that is relevant to the question you created in the previous step. You may search repositories such as Data.gov, UCI Machine Learning, Kaggle, or Scikit-Learn. Find dataset with no less than 10 variables, mostly quantitative.
3. Explain and describe your dataset's variables. List your dependent and independent variables, and identify which scale is used to measure each variable (interval, ordinal, or nominal). Hint: interval is the most appropriate scale for regression analysis.
4. Import all the necessary libraries and load your dataset into a data frame.
5. Use the feature-engine transformers available in the scikit-learn library to feature engineer your dataset variables. Perform the following:
    - Missing data imputation: which transformer(s) did you apply and why?
    - Categorical variable encoding: which transformer(s) did you apply and why?
    - Outliers: which transformer(s) did you apply and why?
    - Discretization: which transformer(s) did you apply and why?
    - Variable transformation: which transformer(s) did you apply and why?
6. Split your dataset into training and testing sets.
7. Import the Pipeline class from the sklearn.pipleline library.
8. Create a pipeline object and pass all the transformers you created in step 5 and a regression model.
9. Fit the pipeline on the training dataset.
10. Make predictions and evaluate the performance of your model using the cross-validation technique. Report the RMSE and R2 values and explain the results.

Tasks 1 and 2

Formulate a prediction question that you want to answer by applying regression modeling. Examples: Prediction Question: How accurately can I predict the price of a house given the values of all the variables?

Search and locate a dataset that is relevant to the question you created in the previous step. You may search repositories such as Data.gov, UCI Machine Learning, Kaggle, or Scikit-Learn. Find dataset with no less than 10 variables, mostly quantitative.

I decided to use a dataset containing data on how many bikes were rented from a bike-sharing service on a particular day. The dataset has features such as the date, season, year, month and day of the week, as well as temperature, humidity, and windspeed data, among others.

Dataset: https://archive.ics.uci.edu/dataset/275/bike+sharing+dataset

Prediction question: How can the number of bikes rented on a particular day be predicted given the values of all the features?

Tasks 3 and 4

Explain and describe your dataset's variables. List your dependent and independent variables, and identify which scale is used to measure each variable (interval, ordinal, or nominal). Hint: interval is the most appropriate scale for regression analysis.

Import all the necessary libraries and load your dataset into a data frame.

In [110…

```
# Import libraries
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_predict
```

In [111…

```
# Load data
df = pd.read_csv("BikeShareDaily.csv")

# View data
df.head()
```

Out[111]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518

Column description (from UCI ML Repository dataset page)

- instant: record index
  - dteday : date.
  - season : season (1: winter, 2: spring, 3: summer, 4: fall).
  - yr : year (0: 2011, 1: 2012).
  - mnth : month (1 to 12).
  - hr : hour (0 to 23).
  - holiday : whether day is holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule).
  - weekday : day of the week.
  - workingday : if day is neither weekend nor holiday is 1, otherwise is 0.
  - weathersit :
    - 1: Clear, Few clouds, Partly cloudy, Partly cloudy.
    - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist.
    - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds.
    - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog.
  - temp : Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-8, t_max=+39 (only in hourly scale).
  - atemp: Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-16, t_max=+50 (only in hourly scale).

- hum: Normalized humidity. The values are divided to 100 (max).
- windspeed: Normalized wind speed. The values are divided to 67 (max).
- casual: count of casual users.
- registered: count of registered users.
- cnt: count of total rental bikes including both casual and registered.

Cleaning the data

In [112…
```
# Rename variables
new_column_names = {"dteday": "date", "yr": "year", "mnth": "month", "weekday": "dayofweek",
                    "workingday": "workday", "weathersit": "weather", "cnt": "count"}
df.rename(columns=new_column_names, inplace=True)
```

Based on the description of the variables, `temp` and `atemp` are similar and thus including both of them in the model could present multicollinearity problems. The unique ID is also not necessary in my opinion. Thus, I have opted to drop `atemp` and `instant`.

In [113…
```
# Drop "atemp" and "instant"
df.drop(columns={"atemp", "instant"}, inplace=True)
```

In [114…
```
# View new column names
df.head()
```

Out[114]:

	date	season	year	month	holiday	dayofweek	workday	weather	temp	hum	windspeed	casual	registered	count
0	2011-01-01	1	0	1	0	6	0	2	0.344167	0.805833	0.160446	331	654	985
1	2011-01-02	1	0	1	0	0	0	2	0.363478	0.696087	0.248539	131	670	801
2	2011-01-03	1	0	1	0	1	1	1	0.196364	0.437273	0.248309	120	1229	1349
3	2011-01-04	1	0	1	0	2	1	1	0.200000	0.590435	0.160296	108	1454	1562
4	2011-01-05	1	0	1	0	3	1	1	0.226957	0.436957	0.186900	82	1518	1600

In [115…
```
# Check for missing values
df.isna().any(axis=1).sum()
```

Out[115]:
```
0
```

The dataset has no missing values, so we are good to go. We will now take a look at descriptive statistics for each variable.

In [116…

```
# Provide descriptive statistics for variables
df.describe().round(3).transpose()
```

Out[116]:

	count	mean	std	min	25%	50%	75%	max
season	731.0	2.497	1.111	1.000	2.000	3.000	3.000	4.000
year	731.0	0.501	0.500	0.000	0.000	1.000	1.000	1.000
month	731.0	6.520	3.452	1.000	4.000	7.000	10.000	12.000
holiday	731.0	0.029	0.167	0.000	0.000	0.000	0.000	1.000
dayofweek	731.0	2.997	2.005	0.000	1.000	3.000	5.000	6.000
workday	731.0	0.684	0.465	0.000	0.000	1.000	1.000	1.000
weather	731.0	1.395	0.545	1.000	1.000	1.000	2.000	3.000
temp	731.0	0.495	0.183	0.059	0.337	0.498	0.655	0.862
hum	731.0	0.628	0.142	0.000	0.520	0.627	0.730	0.972
windspeed	731.0	0.190	0.077	0.022	0.135	0.181	0.233	0.507
casual	731.0	848.176	686.622	2.000	315.500	713.000	1096.000	3410.000
registered	731.0	3656.172	1560.256	20.000	2497.000	3662.000	4776.500	6946.000
count	731.0	4504.349	1937.211	22.000	3152.000	4548.000	5956.000	8714.000

Variable types and scales

Dependent variable: `count`.
Independent variables: `season`, `year`, `month`, `holiday`, `dayofweek`, `workday`, `weather`, `temp`, `hum`, `windspeed`, `casual`, `registered`.

Variable types (interval, ordinal or nominal):

- `season`: ordinal
- `year`: ordinal
- `month`: ordinal
- `holiday`: nominal

- `dayofweek` : ordinal
- `workday` : nominal
- `weather` : nominal
- `temp` : nominal
- `hum` : interval
- `windspeed` : interval
- `casual` : interval
- `registered` : interval
- `count` : interval

Task 5

Use the feature-engine transformers available in the scikit-learn library to feature engineer your dataset variables. Perform the following:

- Missing data imputation: which transformer(s) did you apply and why?
- Categorical variable encoding: which transformer(s) did you apply and why?
- Outliers: which transformer(s) did you apply and why?
- Discretization: which transformer(s) did you apply and why?
- Variable transformation: which transformer(s) did you apply and why?

Because all independent variables are already in numeric format, a categorical variable encoder is not necessary for categorical variables like `season` , `year` , and `month` .

In [118…

```
# Define features to normalize
date = df["date"]
temp = df["temp"]
hum = df["hum"]
windspeed = df["windspeed"]
count = df["count"]
features_to_normalize = df.drop(columns={"date", "temp", "hum", "windspeed", "count"})

# Normalize variables and impute missing values using pipeline
pipeline = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="mean")),
    ("scaler", MinMaxScaler())
```

```
])

# Apply preprocessing
processed_features = pipeline.fit_transform(features_to_normalize)

# Convert processed features to a dataframe
processed_df = pd.DataFrame(processed_features, columns=features_to_normalize.columns)
df = pd.concat([date, temp, hum, windspeed, count, processed_df], axis=1)

# Reorder features so they're in the original order
df = df[["date", "season", "year", "month", "holiday", "dayofweek", "workday", "weather", "temp", "hum", "windspeed",
       "casual", "registered", "count"]]

# Show final dataframe
df.head()
```

Out[118]:

	date	season	year	month	holiday	dayofweek	workday	weather	temp	hum	windspeed	casual	registered	count
0	2011-01-01	0.0	0.0	0.0	0.0	1.000000	0.0	0.5	0.344167	0.805833	0.160446	0.096538	0.091539	985
1	2011-01-02	0.0	0.0	0.0	0.0	0.000000	0.0	0.5	0.363478	0.696087	0.248539	0.037852	0.093849	801
2	2011-01-03	0.0	0.0	0.0	0.0	0.166667	1.0	0.0	0.196364	0.437273	0.248309	0.034624	0.174560	1349
3	2011-01-04	0.0	0.0	0.0	0.0	0.333333	1.0	0.0	0.200000	0.590435	0.160296	0.031103	0.207046	1562
4	2011-01-05	0.0	0.0	0.0	0.0	0.500000	1.0	0.0	0.226957	0.436957	0.186900	0.023474	0.216286	1600

In my pipeline, I applied a simple imputer to fill missing values with the mean. I then used a min-max scaler to normalize the features for modeling. I have opted not to use outlier transformers because the descriptive statistics (specifically min and max) look reasonable for all features. I also chose not to apply any discretization transformers, since outliers are not an issue in this data. Besides imputation and normalization, the variables have not been altered in any way.

Task 6

Split your dataset into training and testing sets.

In [119…
```
# Define features and target
x = df.drop(columns=["date", "count"])
y = df["count"]

# Perform train-test split and show shape of training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
print("Shape of training data: x =", x_train.shape, ", y =", y_train.shape)
print("Shape of testing data: x =", x_test.shape, ", y =", y_test.shape)
```

```
Shape of training data: x = (584, 12) , y = (584,)
Shape of testing data: x = (147, 12) , y = (147,)
```

Task 7

Import the Pipeline class from the sklearn.pipleline library.

In [120…
```
from sklearn.pipeline import Pipeline
```

Task 8

Create a pipeline object and pass all the transformers you created in step 5 and a regression model.

In [121…
```
# Add regression model to the pipeline defined earlier
model = LinearRegression()
pipeline.steps.append(("regression_model", model))
print(pipeline)
```

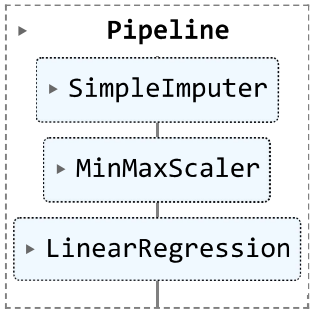
```
Pipeline(steps=[('imputer', SimpleImputer()), ('scaler', MinMaxScaler()),
                ('regression_model', LinearRegression())])
```

Task 9

Fit the pipeline on the training dataset.

In [122…
```
# Fit pipeline to training data
pipeline.fit(x_train, y_train)
```

Out[122]:



Task 10

Make predictions and evaluate the performance of your model using the cross-validation technique. Report the RMSE and R2 values and explain the results.

Making predictions with the model

In [123…

```
# Make predictions on training data
predictions = pipeline.predict(x_train)

# Calculate residuals
residuals = y_train - predictions

# Create new dataframe with actual values, predictions and residuals
results = pd.DataFrame({
    "actual": y_train,
    "predicted": predictions,
    "residual": residuals
})

# Show results
print()
print("Predictions for total bike rentals:")
print()
print(results.head())
print()
print("Mean residual:", results["residual"].mean())
```

```
Predictions for total bike rentals:

     actual  predicted      residual
682    4094     4094.0  0.000000e+00
250    1842     1842.0  6.821210e-13
336    3614     3614.0 -1.364242e-12
260    4274     4274.0  0.000000e+00
543    7335     7335.0 -3.637979e-12

Mean residual: -8.985932241831842e-13
```

At first glance, the residuals are so small that the model looks like it has an overfitting problem with the training data. It will be interesting to see if the evaluation metrics support this hypothesis.

Evaluating the model

In [125…

```python
# Use cross-validation to make predictions
predictions_cv = cross_val_predict(pipeline, x_train, y_train, cv=5)

# Calculate RMSE and R^2
rmse_cv = mean_squared_error(y_train, predictions_cv, squared=False)
r2_cv = r2_score(y_train, predictions_cv)

# Print RMSE and R^2
print(f"Cross-Validation RMSE: {rmse_cv}")
print(f"Cross-Validation R^2: {r2_cv}")
```

```
Cross-Validation RMSE: 3.134398148798729e-12
Cross-Validation R^2: 1.0
```

After performing cross-validation on the model, we see that the RMSE is extremely low and the R^2 is about 1.0, indicating severe overfitting problems with the training data, as I suspected. The model has captured too much of the noise and random fluctuations in the training set, which means its ability to generalize to new data is severely lacking. This could be remedied through further feature engineering, regularization techniques, or alternative cross-validation strategies.

References

ChatGPT. (n.d.). https://chat.openai.com/

GeeksforGeeks. (2024, March 11). ML Underfitting and overfitting. GeeksforGeeks. https://www.geeksforgeeks.org/underfitting-and-

overfitting-in-machine-learning/