Lab_Exercise_1_Simple

July 16, 2025

1 Laboratory Exercise-1: Feature Scaling Methods

AICTE Faculty ID: 1-3241967546
Faculty Name: Milav Jayeshkumar Dabgar
Date: July 16, 2025

1.1 Objective

Apply different feature scaling methods on the Social Network Ads dataset and compare their outputs.

1.2 1. Import Libraries

[1]:
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

print("Libraries imported successfully!")
```

```
Libraries imported successfully!
```

1.3 2. Load Dataset

[2]:
```python
# Load the Social Network Ads dataset
data = pd.read_csv('Social_Network_Ads.csv')

print(f"Dataset shape: {data.shape}")
print("\nFirst 5 rows:")
print(data.head())

print("\nDataset info:")
print(data.info())
```

```
Dataset shape: (400, 3)
```

```
First 5 rows:
   Age  EstimatedSalary  Purchased
0   19            19000          0
1   35            20000          0
2   26            43000          0
3   27            57000          0
4   19            76000          0

Dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Age              400 non-null    int64
 1   EstimatedSalary  400 non-null    int64
 2   Purchased        400 non-null    int64
dtypes: int64(3)
memory usage: 9.5 KB
None
```

1.4 3. Data Preprocessing

[3]:
```
# Select features and target
X = data[['Age', 'EstimatedSalary']]
y = data['Purchased']

print("Features:")
print(X.head())
print("\nTarget:")
print(y.value_counts())
```

```
Features:
   Age  EstimatedSalary
0   19            19000
1   35            20000
2   26            43000
3   27            57000
4   19            76000

Target:
Purchased
0    257
1    143
Name: count, dtype: int64
```

1.5 4. Apply Different Feature Scaling Methods

[4]:
```python
# Original data (no scaling)
print("Original Data:")
print(X.describe())
print("\n" + "="*50)
```

```
Original Data:
              Age  EstimatedSalary
count  400.000000       400.000000
mean    37.655000     69742.500000
std     10.482877     34096.960282
min     18.000000     15000.000000
25%     29.750000     43000.000000
50%     37.000000     70000.000000
75%     46.000000     88000.000000
max     60.000000    150000.000000

==================================================
```

[5]:
```python
# StandardScaler
scaler_std = StandardScaler()
X_standard = scaler_std.fit_transform(X)

print("StandardScaler Output:")
X_std_df = pd.DataFrame(X_standard, columns=['Age_scaled', 'Salary_scaled'])
print(X_std_df.describe())
print("\nFirst 5 rows:")
print(X_std_df.head())
print("\n" + "="*50)
```

```
StandardScaler Output:
         Age_scaled  Salary_scaled
count  4.000000e+02   4.000000e+02
mean  -7.105427e-17  -1.776357e-17
std    1.001252e+00   1.001252e+00
min   -1.877311e+00  -1.607506e+00
25%   -7.550313e-01  -7.852897e-01
50%   -6.256110e-02   7.561451e-03
75%    7.970571e-01   5.361289e-01
max    2.134241e+00   2.356750e+00

First 5 rows:
   Age_scaled  Salary_scaled
0   -1.781797      -1.490046
1   -0.253587      -1.460681
2   -1.113206      -0.785290
3   -1.017692      -0.374182
4   -1.781797       0.183751
```

```
==================================================
```

[6]:
```
# MinMaxScaler
scaler_mm = MinMaxScaler()
X_minmax = scaler_mm.fit_transform(X)

print("MinMaxScaler Output:")
X_mm_df = pd.DataFrame(X_minmax, columns=['Age_scaled', 'Salary_scaled'])
print(X_mm_df.describe())
print("\nFirst 5 rows:")
print(X_mm_df.head())
print("\n" + "="*50)
```

```
MinMaxScaler Output:
       Age_scaled  Salary_scaled
count  400.000000     400.000000
mean     0.467976       0.405500
std      0.249592       0.252570
min      0.000000       0.000000
25%      0.279762       0.207407
50%      0.452381       0.407407
75%      0.666667       0.540741
max      1.000000       1.000000

First 5 rows:
   Age_scaled  Salary_scaled
0    0.023810       0.029630
1    0.404762       0.037037
2    0.190476       0.207407
3    0.214286       0.311111
4    0.023810       0.451852

==================================================
```

[7]:
```
# RobustScaler
scaler_rob = RobustScaler()
X_robust = scaler_rob.fit_transform(X)

print("RobustScaler Output:")
X_rob_df = pd.DataFrame(X_robust, columns=['Age_scaled', 'Salary_scaled'])
print(X_rob_df.describe())
print("\nFirst 5 rows:")
print(X_rob_df.head())
print("\n" + "="*50)
```

```
RobustScaler Output:
       Age_scaled  Salary_scaled
count  400.000000     400.000000
```

```
mean      0.040308      -0.005722
std       0.645100       0.757710
min      -1.169231      -1.222222
25%      -0.446154      -0.600000
50%       0.000000       0.000000
75%       0.553846       0.400000
max       1.415385       1.777778

First 5 rows:
   Age_scaled  Salary_scaled
0   -1.107692      -1.133333
1   -0.123077      -1.111111
2   -0.676923      -0.600000
3   -0.615385      -0.288889
4   -1.107692       0.133333

==================================================
```

1.6 5. Train-Test Split and Model Comparison

[8]:
```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Test different scaling methods
scalers = {
    'No Scaling': None,
    'StandardScaler': StandardScaler(),
    'MinMaxScaler': MinMaxScaler(),
    'RobustScaler': RobustScaler()
}

results = []

for name, scaler in scalers.items():
    if scaler is None:
        X_train_scaled = X_train
        X_test_scaled = X_test
    else:
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

    # Train model
    model = LogisticRegression(random_state=42)
    model.fit(X_train_scaled, y_train)

    # Predict and calculate accuracy
```

```python
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)

    results.append({'Scaling Method': name, 'Accuracy': accuracy})
    print(f"{name}: {accuracy:.4f}")

# Show results
results_df = pd.DataFrame(results)
print("\nResults Summary:")
print(results_df)
```

```
No Scaling: 0.8875
StandardScaler: 0.8625
MinMaxScaler: 0.8750
RobustScaler: 0.8625

Results Summary:
   Scaling Method  Accuracy
0      No Scaling    0.8875
1  StandardScaler    0.8625
2    MinMaxScaler    0.8750
3    RobustScaler    0.8625
```

1.7 6. Conclusion

1.7.1 Experimental Results Summary:

From our feature scaling experiments on the Social Network Ads dataset, we obtained the following accuracy results:

Scaling Method	Accuracy
No Scaling	**88.75%**
MinMaxScaler	87.50%
StandardScaler	86.25%
RobustScaler	86.25%

1.7.2 Key Findings:

1. **No scaling performed best** (88.75%) - This indicates that the original features (Age and Salary) were already on reasonable scales for logistic regression.
2. **Feature scaling characteristics observed:**
   - **StandardScaler**: Successfully normalized data to mean 0, std 1
   - **MinMaxScaler**: Scaled all values to [0,1] range
   - **RobustScaler**: Used median and IQR, robust to outliers
3. **All scaling methods** successfully transformed the data while preserving the underlying relationships between features and target variable.

1.7.3 Learning Outcomes Achieved:

- Applied multiple feature scaling techniques
- Compared scaling method outputs and transformations
- Evaluated model performance impact of different scaling approaches
- Demonstrated that scaling choice depends on data characteristics and algorithm

Conclusion: While feature scaling is often crucial for machine learning algorithms, this experiment shows that some datasets and algorithms (like this logistic regression on Social Network Ads) may work well even without scaling, emphasizing the importance of experimental validation.