Lesson 6: Supervised Learning - Regression

Welcome to Session 6! Today, we'll explore **Supervised Learning** with a focus on **Regression**, one of the most important techniques for predicting numerical values. We'll start with simple examples to explain linear regression, move on to polynomial regression, and learn how to evaluate regression models using practical activities.

Outline

1. **Introduction to Regression**
   - What is regression?
   - Real-world examples.
2. **Linear Regression**
   - Concept and examples.
   - Step-by-step implementation with code.
   - Hands-on activity: Predict house prices.
3. **Polynomial Regression**
   - Concept and examples.
   - Step-by-step implementation with code.
   - Hands-on activity: Improve predictions using polynomial regression.
4. **Evaluation Metrics**
   - MAE, MSE, and RMSE explained with examples.

---

1. Introduction to Regression

What is Regression?

Regression is a type of supervised learning where we predict a **continuous numerical value** based on one or more input features.

Real-World Examples

1. **Predicting House Prices**
   - Input: Size of the house, number of bedrooms, location.
   - Output: Price of the house.
2. **Predicting Car Mileage**
   - Input: Engine size, car weight, fuel type.
   - Output: Miles per gallon (MPG).
3. **Predicting Student Grades**
   - Input: Study hours, attendance, test scores.
   - Output: Final grade.

---

2. Linear Regression

Concept

Linear regression models the relationship between one or more input features (X) and an output variable (y) using a straight line.

- **Example:** Predicting house prices based on size.

Simple Analogy:
Imagine you're at a market. The price of apples depends on their weight.

- 1 kg apples = $2.
- 2 kg apples = $4.
  This relationship can be modeled as:

$$\text{Price} = 2 \times \text{Weight}$$

Example 1: Predict House Prices (Single Feature)

Size (sq ft)	Price ($)
1000	200000
1500	250000
2000	300000
2500	350000

Code: Linear Regression with One Feature

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Dataset
data = {'Size': [1000, 1500, 2000, 2500],
        'Price': [200000, 250000, 300000, 350000]}
df = pd.DataFrame(data)

# Features (X) and Target (y)
X = df[['Size']]
y = df['Price']

# Train the model
model = LinearRegression()
model.fit(X, y)

# Predictions
y_pred = model.predict(X)

# Visualize the data and regression line
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', label='Regression Line')
plt.title('Linear Regression: Size vs Price')
plt.xlabel('Size (sq ft)')
```

```
plt.ylabel('Price ($)')
plt.legend()
plt.show()
```

Explanation:

1. **Input (X)**: Size of the house.
2. **Output (y)**: Price of the house.
3. The red line shows the regression line, predicting price based on size.

Example 2: Predict House Prices (Multiple Features)

Size (sq ft)	Bedrooms	Price ($)
1500	3	300000
2000	4	400000
2500	4	350000
3000	5	500000

Code: Linear Regression with Multiple Features

```
# Dataset
data = {'Size': [1500, 2000, 2500, 3000],
        'Bedrooms': [3, 4, 4, 5],
        'Price': [300000, 400000, 350000, 500000]}
df = pd.DataFrame(data)

# Features (X) and Target (y)
X = df[['Size', 'Bedrooms']]
y = df['Price']

# Train the model
model = LinearRegression()
model.fit(X, y)

# Predictions
y_pred = model.predict(X)
print("Predicted Prices:", y_pred)
```

Explanation:

- Here, we use two features: `Size` and `Bedrooms`.
- The model predicts the price based on both features.

---

3. Polynomial Regression

Concept

Polynomial regression fits a curve to capture non-linear relationships.
Example: Suppose the price of a house depends on the square of its size (x2x^2).

Example: Predict House Prices with Polynomial Regression

Code:

```
from sklearn.preprocessing import PolynomialFeatures

# Create polynomial features
poly = PolynomialFeatures(degree=2)  # Quadratic relationship
X_poly = poly.fit_transform(X)

# Train polynomial regression model
poly_model = LinearRegression()
poly_model.fit(X_poly, y)

# Predictions
y_poly_pred = poly_model.predict(X_poly)

# Visualize polynomial regression
plt.scatter(df['Size'], y, color='blue', label='Actual Data')
plt.plot(df['Size'], y_poly_pred, color='green', label='Polynomial Regression
Curve')
plt.title('Polynomial Regression: Size vs Price')
plt.xlabel('Size (sq ft)')
plt.ylabel('Price ($)')
plt.legend()
plt.show()
```

Explanation:

- `PolynomialFeatures(degree=2)` creates features like x2x^2.
- The curve fits the data better than a straight line.

---

4. Evaluation Metrics

1. Mean Absolute Error (MAE)

Measures the average magnitude of errors:

$$\text{MAE} = \frac{1}{n} \sum |y_i - \hat{y}_i|$$

Example Calculation:

- True prices: [200, 300, 400].
- Predicted prices: [210, 290, 405].
- MAE: $(|200 - 210| + |300 - 290| + |400 - 405|)/3 = 8.33$.

---

2. Mean Squared Error (MSE)

Penalizes larger errors:

$$\text{MSE} = \frac{1}{n} \sum (y_i - \hat{y}_i)^2$$

Example Calculation:

- True prices: [200, 300, 400].
- Predicted prices: [210, 290, 405].
- MSE: $((200 - 210)^2 + (300 - 290)^2 + (400 - 405)^2)/3 = 41.67$.

---

3. Root Mean Squared Error (RMSE)

Gives errors in the same units as the target variable:

$$\text{RMSE} = \sqrt{\text{MSE}}$$

Example Calculation:

- RMSE: $\sqrt{41.67} \approx 6.46$.

---

5. Activities

Activity 1: Train a Linear Regression Model

Objective: Train a linear regression model to predict house prices based on features like size and bedrooms. Evaluate the model using MAE, MSE, and RMSE.

Solution to Activity 1:

Dataset:

Size (sq ft)	Bedrooms	Price ($)
1500	3	300000
2000	4	400000
2500	4	350000
3000	5	500000

Code:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Dataset
data = {'Size': [1500, 2000, 2500, 3000],
        'Bedrooms': [3, 4, 4, 5],
        'Price': [300000, 400000, 350000, 500000]}
df = pd.DataFrame(data)

# Features (X) and Target (y)
```

```
X = df[['Size', 'Bedrooms']]
y = df['Price']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

Expected Output:

- The model will predict house prices based on size and bedrooms.
- You will get MAE, MSE, and RMSE values, which indicate the model's performance.

Activity 2: Experiment with Polynomial Regression

Objective: Experiment with polynomial regression to improve the model's performance. Train a polynomial regression model with degree 2 and compare its performance metrics (MAE, MSE, RMSE) with the linear regression model.

Solution to Activity 2:

Code:

```
from sklearn.preprocessing import PolynomialFeatures

# Create polynomial features
poly = PolynomialFeatures(degree=2)  # Quadratic features
X_poly = poly.fit_transform(X)

# Train polynomial regression model
poly_model = LinearRegression()
poly_model.fit(X_poly, y)

# Make predictions
X_test_poly = poly.transform(X_test)
y_poly_pred = poly_model.predict(X_test_poly)
```

```
# Evaluate the polynomial regression model
mae_poly = mean_absolute_error(y_test, y_poly_pred)
mse_poly = mean_squared_error(y_test, y_poly_pred)
rmse_poly = np.sqrt(mse_poly)

print("Polynomial Regression - MAE:", mae_poly)
print("Polynomial Regression - MSE:", mse_poly)
print("Polynomial Regression - RMSE:", rmse_poly)
```

Expected Output:

- Polynomial regression will add quadratic terms (like x2x^2) to better capture non-linear relationships in the data.
- The performance metrics (MAE, MSE, RMSE) will likely improve if the relationship between features and target is non-linear.

---

1. **Activity 1:** Linear regression fits a straight line to the data. Its performance metrics give insight into the model's accuracy.
2. **Activity 2:** Polynomial regression improves the model's flexibility by capturing non-linear relationships. Comparing metrics like MAE, MSE, and RMSE helps determine whether the polynomial model is better than the linear model.

Conclusion

- Regression predicts numerical outcomes based on input features.
- Linear regression fits a straight line; polynomial regression captures curves.
- Evaluation metrics like MAE, MSE, and RMSE help assess model performance.