Supervised Machine Learning: Regression
Personal Detailed Notebook

Contents

1 Module 1: Introduction to Supervised Learning and Regression

1.1 Overview

Supervised machine learning models a mapping from input features x to continuous outputs y using labeled data. The goal is to predict y given new x.

1.2 Model Representation

$$y = f(x) + \varepsilon$$

where ε is noise/error.

Linear regression assumes:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

1.3 Example: Predict House Prices from Square Footage

1.4 Python Code

```python
// Import libraries
import numpy as np
from sklearn.linear_model import LinearRegression

# Training data: square footage (X) and house price in thousands (y)
X = np.array([[1000], [1200], [1500], [1700]])
y = np.array([150, 200, 240, 300])

# Create and train the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict price for a 1700 sqft house
prediction = model.predict(np.array([[1700]]))
print(f"Predicted price: {prediction[0]:.1f}k")
```

Listing 1: Linear Regression Example

1.5 Visual Illustration

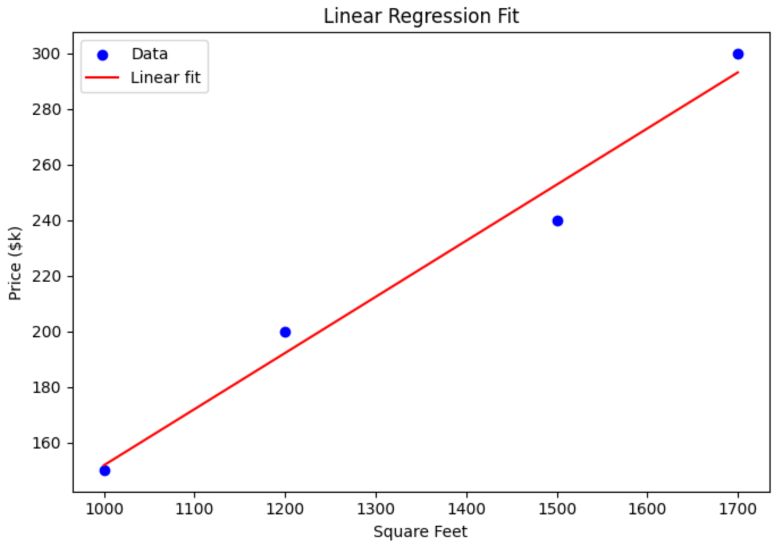

Figure 1: Linear regression fit on house price data

2 Module 2: Model Evaluation, Error Metrics, and Data Splitting

2.1 Error Metrics

Common error metrics:

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

2.2 Train/Test Data Splitting

Split data into training and testing set to evaluate generalization.

2.3 Python Code

```
// Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Split with 25% test size
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=1)

# Train model on training set
model.fit(X_train, y_train)

# Predict on test set
```

```
y_pred = model.predict(X_test)

# Calculate MSE on test set
mse = mean_squared_error(y_test, y_pred)
print(f"Test MSE: {mse:.2f}")
```

Listing 2: Data Split and Evaluation

2.4 Visual Illustration

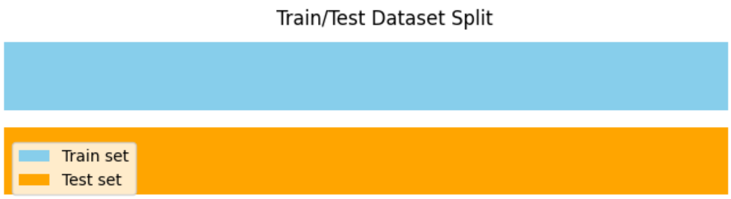

Figure 2: Dataset split into training and testing parts

3 Module 3: Polynomial Regression and Overfitting

3.1 Concept

Polynomial regression models include powers of features to capture nonlinearities:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_d x^d$$

High degree can cause overfitting.

3.2 Python Code

```
// Import library for polynomial features
from sklearn.preprocessing import PolynomialFeatures

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

# Train model on polynomial features
model.fit(X_poly, y)

# Predict for 1700 sqft
prediction = model.predict(poly.transform(np.array([[1700]])))
print(f"Predicted price (quadratic): {prediction[0]:.1f}k")
```

Listing 3: Polynomial Regression

3.3 Visual Illustration

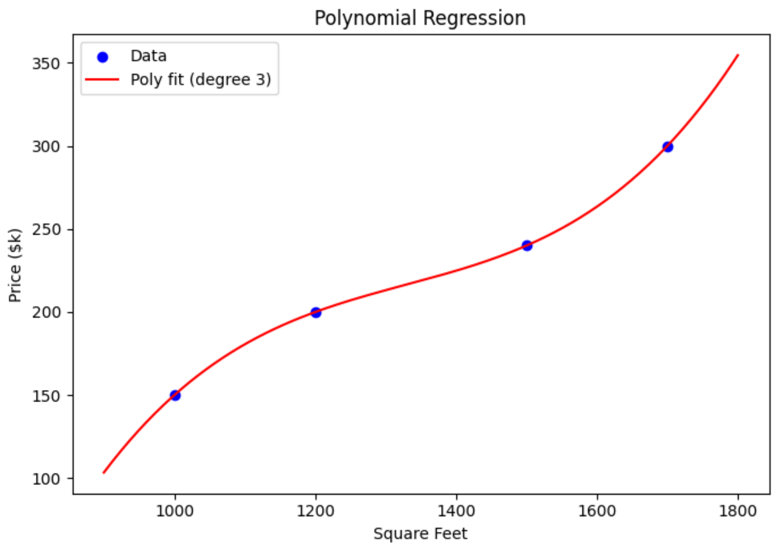

Figure 3: Polynomial regression curve fit

4 Module 4: Regularization — Ridge and Lasso

4.1 Overview

Regularization penalizes complexity to reduce overfitting.

- Ridge regression (L2 penalty):

$$J(\theta) = MSE + \lambda \sum_{j=1}^{n} \theta_j^2$$

- Lasso regression (L1 penalty):

$$J(\theta) = MSE + \lambda \sum_{j=1}^{n} |\theta_j|$$

λ controls regularization strength.

4.2 Python Code

```
// Import regression models
from sklearn.linear_model import Ridge, Lasso

# Ridge regression with alpha=1.0
ridge = Ridge(alpha=1.0)
ridge.fit(X, y)
print("Ridge coefficients:", ridge.coef_)

# Lasso regression with alpha=1.0
lasso = Lasso(alpha=1.0)
lasso.fit(X, y)
print("Lasso coefficients:", lasso.coef_)
```

Listing 4: Ridge and Lasso Regression

4.3 Visual Illustration

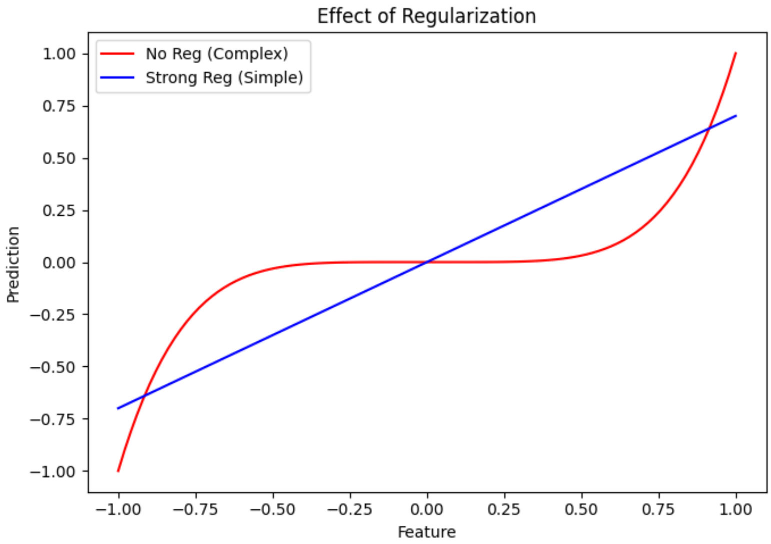

Figure 4: Regularization controlling model complexity

5 Module 5: Model Selection, Learning Curves, and Cross Validation

5.1 Concept

Use cross validation and learning curves to select and assess models. Learning curves plot training and validation error as training set size increases.

5.2 Python Code

```
// Import needed sklearn functions
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt

train_sizes, train_scores, test_scores = learning_curve(
    LinearRegression(), X, y, cv=5,
    train_sizes=np.linspace(0.2, 1.0, 5))

train_rmse = np.sqrt(1 - train_scores.mean(axis=1))
test_rmse = np.sqrt(1 - test_scores.mean(axis=1))

plt.plot(train_sizes, train_rmse, label='Train RMSE')
plt.plot(train_sizes, test_rmse, label='Test RMSE')
plt.xlabel('Training set size')
plt.ylabel('RMSE')
plt.title('Learning Curve')
plt.legend()
plt.tight_layout()
plt.savefig('learning_curve.png')
plt.close()
```

Listing 5: Learning Curve

5.3 Visual Illustration

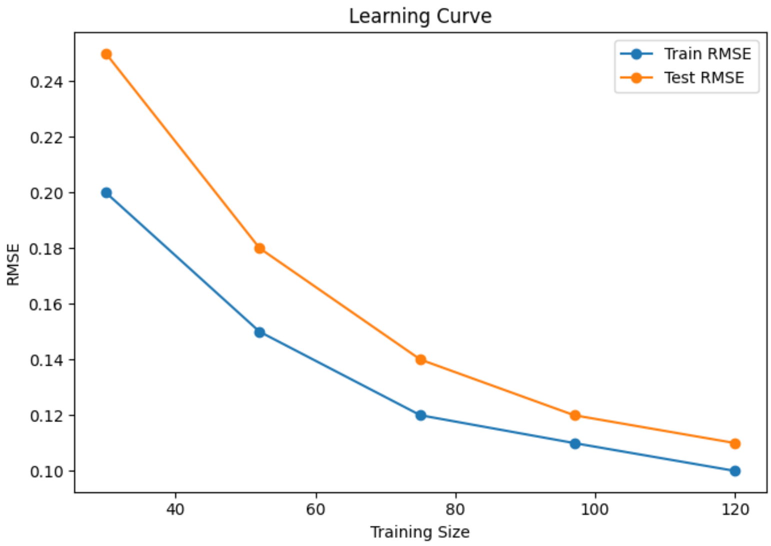

Figure 5: Learning curve showing train and test RMSE