# Regression Models and Evaluation Techniques

July 3, 2025

## 1 Introduction to Regression

### 1.1 What is Regression?

Regression is the task of predicting continuous numerical values rather than discrete categories. It's one of the fundamental supervised learning tasks in machine learning, focusing on understanding relationships between variables.

### 1.2 Real-World Examples

- **House price prediction:** Estimating property values based on features
- **Stock price forecasting:** Predicting future market values
- **Sales forecasting:** Estimating future revenue
- **Temperature prediction:** Weather forecasting models
- **Medical dosage:** Determining optimal medication amounts

**Today's Goals:**

1. Learn 5 powerful regression models
2. Master evaluation techniques for regression
3. Understand when to use each approach

## 2 Data Preparation

### 2.1 Setting Up Practice Data

We'll create synthetic regression data to demonstrate each model and evaluation technique.

```
import numpy as np
import pandas as pd
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Create synthetic regression dataset
X, y = make_regression(n_samples=1000, n_features=10, noise=0.1, random_state
    =42)

# Split data (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    random_state=42)
```

```

# Scale features for models that need it
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(f"Training set size: {X_train.shape}")
print(f"Test set size: {X_test.shape}")
```

Listing 1: Data Setup for Regression

# 3 Regression Models

## 3.1 Model 1: Linear Regression

**How it works:** Finds the best-fitting straight line (or hyperplane) through the data by minimizing the sum of squared residuals.

**Mathematical foundation:** $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \epsilon$

**Best for:** Linear relationships, interpretable coefficients

**Limitations:** Assumes linear relationships, sensitive to outliers

**Example use:** House price prediction based on size, location

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Create and train model
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)

# Make predictions
y_pred = linear_reg.predict(X_test)

# Evaluate performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Linear Regression Results:")
print(f"MSE: {mse:.2f}")
print(f"R  Score: {r2:.3f}")
```

Listing 2: Linear Regression Implementation

## 3.2 Model 2: Ridge Regression

**How it works:** Linear regression with L2 regularization that adds a penalty term to prevent overfitting by shrinking coefficients.

**Mathematical foundation:** Minimizes: $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{p} \beta_j^2$

**Best for:** High-dimensional data, multicollinearity issues

**Limitations:** Doesn't perform feature selection

**Example use:** Gene expression analysis, text analysis

```python
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

# Create model with regularization
ridge_reg = Ridge()

# Tune hyperparameter alpha
param_grid = {'alpha': [0.1, 1.0, 10.0, 100.0]}
ridge_cv = GridSearchCV(ridge_reg, param_grid, cv=5, scoring='
    neg_mean_squared_error')
ridge_cv.fit(X_train_scaled, y_train)

# Best model predictions
y_pred_ridge = ridge_cv.predict(X_test_scaled)

# Evaluate performance
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f"Ridge Regression Results:")
print(f"Best alpha: {ridge_cv.best_params_['alpha']}")
print(f"MSE: {mse_ridge:.2f}")
print(f" R   Score: {r2_ridge:.3f}")
```

Listing 3: Ridge Regression Implementation

### 3.3 Model 3: Lasso Regression

**How it works:** Linear regression with L1 regularization that can shrink coefficients to exactly zero, performing automatic feature selection.

**Mathematical foundation:** Minimizes: $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^{p} |\beta_j|$

**Best for:** Feature selection, sparse models

**Limitations:** Can be unstable with correlated features

**Example use:** Biomarker discovery, economics modeling

```python
from sklearn.linear_model import Lasso

# Create model with L1 regularization
lasso_reg = Lasso()

# Tune hyperparameter alpha
param_grid = {'alpha': [0.01, 0.1, 1.0, 10.0]}
lasso_cv = GridSearchCV(lasso_reg, param_grid, cv=5, scoring='
    neg_mean_squared_error')
lasso_cv.fit(X_train_scaled, y_train)

# Best model predictions
y_pred_lasso = lasso_cv.predict(X_test_scaled)

# Evaluate performance
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

print(f"Lasso Regression Results:")
print(f"Best alpha: {lasso_cv.best_params_['alpha']}")
print(f"MSE: {mse_lasso:.2f}")
print(f" R   Score: {r2_lasso:.3f}")
```

```
print(f"Features selected: {np.sum(lasso_cv.best_estimator_.coef_ != 0)}")
```

Listing 4: Lasso Regression Implementation

### 3.4 Model 4: Random Forest Regression

**How it works:** Ensemble of decision trees that averages predictions from multiple trees, each trained on different subsets of data and features.

**Best for:** Non-linear relationships, robust performance

**Limitations:** Less interpretable, can overfit with small datasets

**Example use:** Real estate valuation, demand forecasting

```
from sklearn.ensemble import RandomForestRegressor

# Create random forest model
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(X_train, y_train)

# Make predictions
y_pred_rf = rf_reg.predict(X_test)

# Evaluate performance
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest Regression Results:")
print(f"MSE: {mse_rf:.2f}")
print(f"R  Score: {r2_rf:.3f}")

# Feature importance
feature_importance = rf_reg.feature_importances_
print(f"Top 3 most important features: {np.argsort(feature_importance)[-3:]}")
```

Listing 5: Random Forest Regression Implementation

### 3.5 Model 5: Support Vector Regression (SVR)

**How it works:** Finds a function that deviates from actual values by at most $\epsilon$ while being as flat as possible.

**Best for:** High-dimensional data, non-linear relationships (with kernels)

**Limitations:** Sensitive to feature scaling, computationally expensive

**Example use:** Financial modeling, time series prediction

```
from sklearn.svm import SVR

# Create SVR model with RBF kernel
svr_reg = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1)
svr_reg.fit(X_train_scaled, y_train)

# Make predictions
y_pred_svr = svr_reg.predict(X_test_scaled)

# Evaluate performance
```

```
mse_svr = mean_squared_error(y_test, y_pred_svr)
r2_svr = r2_score(y_test, y_pred_svr)

print(f"Support Vector Regression Results:")
print(f"MSE: {mse_svr:.2f}")
print(f"R   Score: {r2_svr:.3f}")
```

Listing 6: Support Vector Regression Implementation

# 4 Evaluation Techniques

## 4.1 Common Regression Metrics

### 4.1.1 Mean Squared Error (MSE)

**Formula:** $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

**Interpretation:** Average of squared differences between actual and predicted values. Lower is better.

### 4.1.2 Root Mean Squared Error (RMSE)

**Formula:** $RMSE = \sqrt{MSE}$

**Interpretation:** Same units as target variable, easier to interpret than MSE.

### 4.1.3 Mean Absolute Error (MAE)

**Formula:** $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$

**Interpretation:** Average absolute difference, less sensitive to outliers than MSE.

### 4.1.4 R-squared ($R^2$)

**Formula:** $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$

**Interpretation:** Proportion of variance explained by the model. Range: 0 to 1 (higher is better).

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

def evaluate_regression_model(y_true, y_pred, model_name):
    """
    Comprehensive evaluation of regression model performance
    """
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)

    print(f"\n{model_name} Performance:")
    print(f"{'='*50}")
    print(f"MSE:  {mse:.4f}")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAE:  {mae:.4f}")
    print(f"R  :   {r2:.4f}")

    return {'MSE': mse, 'RMSE': rmse, 'MAE': mae, 'R2': r2}

# Example usage for all models
results = {}
```

```
results['Linear'] = evaluate_regression_model(y_test, y_pred, "Linear Regression")
results['Ridge'] = evaluate_regression_model(y_test, y_pred_ridge, "Ridge Regression")
results['Lasso'] = evaluate_regression_model(y_test, y_pred_lasso, "Lasso Regression")
results['Random Forest'] = evaluate_regression_model(y_test, y_pred_rf, "Random Forest")
results['SVR'] = evaluate_regression_model(y_test, y_pred_svr, "Support Vector Regression")
```

Listing 7: Comprehensive Evaluation Function

## 4.2 Cross-Validation

Cross-validation provides more robust performance estimates by using multiple train/test splits.

```
from sklearn.model_selection import cross_val_score

def cross_validate_model(model, X, y, cv=5, scoring='neg_mean_squared_error'):
    """
    Perform cross-validation for regression model
    """
    scores = cross_val_score(model, X, y, cv=cv, scoring=scoring)

    print(f"Cross-Validation Results (CV={cv}):")
    print(f"Mean MSE: {-scores.mean():.4f}")
    print(f"Std MSE:  {scores.std():.4f}")
    print(f"95% CI:   [{-scores.mean() - 1.96*scores.std():.4f}, "
          f"{-scores.mean() + 1.96*scores.std():.4f}]")

    return scores

# Example: Cross-validate Random Forest
rf_scores = cross_validate_model(RandomForestRegressor(n_estimators=100, random_state=42),
                                 X_train, y_train)
```

Listing 8: Cross-Validation for Regression

## 4.3 Residual Analysis

Analyzing residuals helps identify model assumptions violations and potential improvements.

```
import matplotlib.pyplot as plt

def plot_residuals(y_true, y_pred, model_name):
    """
    Create residual plots for regression analysis
    """
    residuals = y_true - y_pred

    fig, axes = plt.subplots(1, 2, figsize=(12, 5))

    # Residuals vs Predicted
    axes[0].scatter(y_pred, residuals, alpha=0.6)
    axes[0].axhline(y=0, color='r', linestyle='--')
    axes[0].set_xlabel('Predicted Values')
    axes[0].set_ylabel('Residuals')
    axes[0].set_title(f'{model_name}: Residuals vs Predicted')

    # Q-Q plot for normality
```

```
    from scipy import stats
    stats.probplot(residuals, dist="norm", plot=axes[1])
    axes[1].set_title(f'{model_name}: Q-Q Plot')

    plt.tight_layout()
    plt.show()

    # Statistical tests
    print(f"\nResidual Analysis for {model_name}:")
    print(f"Mean of residuals: {np.mean(residuals):.6f}")
    print(f"Std of residuals:  {np.std(residuals):.4f}")

# Example usage
plot_residuals(y_test, y_pred_rf, "Random Forest")
```

Listing 9: Residual Analysis

# 5 Model Comparison

| Model | Strengths | Weaknesses | Best Use Cases |
|---|---|---|---|
| Linear Regression | Simple, interpretable, fast | Assumes linearity, sensitive to outliers | Linear relationships, baseline models |
| Ridge Regression | Handles multicollinearity, prevents overfitting | Doesn't select features | High-dimensional data, correlated features |
| Lasso Regression | Feature selection, sparse models | Unstable with correlated features | Feature selection, interpretable models |
| Random Forest | Handles nonlinearity, robust | Less interpretable, can overfit | Complex relationships, robust predictions |
| SVR | Flexible with kernels, robust to outliers | Computationally expensive, requires scaling | High-dimensional data, non-linear patterns |

Table 1: Comparison of Regression Models

# 6 Evaluation Metrics Comparison

# 7 Best Practices

## 7.1 Model Selection Guidelines

1. **Start simple:** Begin with linear regression as baseline
2. **Consider data size:** Use regularized models for small datasets
3. **Check assumptions:** Verify linearity, normality, homoscedasticity
4. **Handle multicollinearity:** Use Ridge regression when features are correlated
5. **Feature selection:** Use Lasso when you need interpretable models

| Metric | Formula | Interpretation | When to Use |
|---|---|---|---|
| MSE | $\frac{1}{n}\sum(y_i - \hat{y}_i)^2$ | Penalizes large errors heavily | When large errors are costly |
| RMSE | $\sqrt{MSE}$ | Same units as target | General purpose, interpretable |
| MAE | $\frac{1}{n}\sum\|y_i - \hat{y}_i\|$ | Robust to outliers | When outliers present |
| $R^2$ | $1 - \frac{SS_{res}}{SS_{tot}}$ | Proportion of variance explained | Model comparison, goodness of fit |
| MAPE | $\frac{100}{n}\sum\frac{\|y_i - \hat{y}_i\|}{y_i}$ | Percentage error | Relative performance |

Table 2: Regression Evaluation Metrics

6. **Non-linear relationships:** Try Random Forest or SVR with appropriate kernels

### 7.2 Evaluation Best Practices

1. **Multiple metrics:** Don't rely on a single metric
2. **Cross-validation:** Use k-fold CV for robust estimates
3. **Residual analysis:** Check model assumptions
4. **Domain context:** Consider business/domain-specific requirements
5. **Overfitting check:** Compare training vs. validation performance

## 8 Conclusion

Regression modeling requires careful consideration of data characteristics, model assumptions, and evaluation metrics. Start with simple linear models to establish baselines, then explore more complex approaches as needed. Always validate your models using appropriate metrics and cross-validation techniques, and don't forget to analyze residuals to ensure model assumptions are met.

The key to successful regression modeling lies in understanding your data, choosing appropriate models for your specific use case, and thoroughly evaluating performance using multiple metrics and validation techniques.