CS109A: Introduction to Data Science
Lecture 05: Linear Regression

Harvard University

Fall 2024

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 05: Linear Regression
- **Instructor:** Pavlos Protopapas
- **Objective:** Master simple and multiple linear regression, understand the normal equation, interpret coefficients, and handle practical issues like scaling and collinearity

Key Summary

This lecture introduces **Linear Regression**—the foundational model for all of machine learning. We start with **Simple Linear Regression** (one predictor), derive the optimal coefficients using calculus, then extend to **Multiple Linear Regression** (many predictors) using matrix notation. We learn to **interpret coefficients**, understand the importance of **feature scaling**, recognize **collinearity** problems, and handle **categorical variables**. Linear regression is the "first principles" model that helps you understand every other ML algorithm.

Contents

1 Key Terminology

Table 1: *Linear Regression Terminology*

Term	**Symbol**	**Description**
Response Variable	Y or y	The outcome we're predicting
Predictor Variable	X or x	Input used for prediction
Intercept	β_0	Value of Y when all Xs are zero
Coefficient/Slope	$\beta_1, \beta_2, ...$	Effect of each predictor on Y
Residual	$r_i = y_i - \hat{y}_i$	Error for one observation
Loss Function	$L(\beta)$	Measures how wrong the model is
MSE	$\frac{1}{n}\sum(y_i - \hat{y}_i)^2$	Mean Squared Error
Normal Equation	$\hat{\beta} = (X^T X)^{-1} X^T y$	Closed-form solution
Design Matrix	X	Matrix of all predictors ($n \times (p+1)$)
Feature Scaling	—	Normalizing predictors to similar scales
Collinearity	—	When predictors are correlated with each other

2 Why Linear Regression?

2.1 The Foundation of All Machine Learning

You might wonder: "Why study something so simple when neural networks exist?"

Key Information

Linear Regression is the Rosetta Stone of ML
Even the most complex models (neural networks, transformers, GPT) follow the same pattern:

1. **Define a model** (structure/hypothesis)
2. **Define a loss function** (measure of error)
3. **Minimize the loss** (find optimal parameters)

Linear regression is the simplest case where you can see this pattern clearly. Master it, and every other algorithm becomes easier to understand.

2.2 Interpretability: The Killer Feature

Unlike kNN ("the neighbors said so"), linear regression tells you **exactly how each variable affects the outcome**.

Example:

kNN vs Linear Regression **Question**: "What happens if we double the TV advertising budget?"
kNN answer: "Uh... let me find similar data points and recalculate..."
Linear Regression answer: "The coefficient is 0.05, so every \$1,000 increase in TV budget increases sales by 50 units. Doubling from \$100k to \$200k should increase sales by approximately 5,000 units."
Linear regression gives you **actionable insights**.

3 Simple Linear Regression (SLR)

3.1 The Model

We assume a **linear relationship** between one predictor X and response Y:

Definition:

Simple Linear Regression

$$Y = \beta_0 + \beta_1 X + \epsilon$$

- β_0: **Intercept** — value of Y when $X = 0$
- β_1: **Slope** — change in Y for a 1-unit change in X
- ϵ: **Error** — everything the model can't explain

Our prediction (without error) is:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$

3.2 Finding the Best Line: Loss Function

Among all possible lines, which is "best"? The one that's **closest to all data points**.

3.2.1 Residuals

Definition:

Residual The **residual** for observation i is the vertical distance from the point to the line:

$$r_i = y_i - \hat{y}_i = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)$$

A good model has small residuals.

3.2.2 Mean Squared Error (MSE)

Definition:

MSE Loss Function

$$L(\beta_0, \beta_1) = \text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$$

Warning

Why Square the Residuals?

If we just summed residuals, positive errors ($y > \hat{y}$) and negative errors ($y < \hat{y}$) would cancel out. A model could have huge errors that sum to zero!

Squaring ensures:

1. All errors are positive (no cancellation)

2. Large errors are penalized more heavily ($10^2 = 100$ vs $2^2 = 4$)
3. The function is differentiable everywhere (nice for optimization)

3.3 The Three Steps of Machine Learning

Important:

The Universal ML Recipe

1. **Define the Model**: $\hat{Y} = \beta_0 + \beta_1 X$ (assume a linear relationship)
2. **Define the Loss**: $L = \frac{1}{n}\sum(y_i - \hat{y}_i)^2$ (MSE measures error)
3. **Minimize the Loss**: Find $\hat{\beta}_0, \hat{\beta}_1$ that minimize L

This pattern applies to almost every ML algorithm!

- Neural networks: Same idea, but model is more complex
- ChatGPT: Define transformer model, define cross-entropy loss, minimize with backpropagation

3.4 Minimizing MSE: The Calculus

How do we find the β values that minimize MSE?

Analogy:

Finding the Bottom of a Bowl Imagine MSE as a 3D bowl where:

- x-axis = β_0
- y-axis = β_1
- z-axis = MSE value

The optimal point is at the **bottom of the bowl** where the surface is flat (slope = 0).

Mathematically: Set the **partial derivatives** to zero.

3.4.1 Setting Derivatives to Zero

To find the minimum, we set:

$$\frac{\partial L}{\partial \beta_0} = 0 \quad \text{and} \quad \frac{\partial L}{\partial \beta_1} = 0$$

3.4.2 The Normal Equation (Closed-Form Solution)

Solving these equations gives us:

Simple Linear Regression: Optimal Coefficients

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

where $\bar{x}$ and $\bar{y}$ are the means of X and Y.

Key Information

What `.fit()` Actually Does

When you call `model.fit(X, y)` in sklearn, it computes these formulas using your data. That's it—no magic, just algebra!

3.5 Implementation in Python

```python
from sklearn.linear_model import LinearRegression
import pandas as pd

# Load data
df = pd.read_csv('advertising.csv')

# Prepare X and y
X = df[['TV']]          # Double brackets: returns DataFrame (required by sklearn)
y = df['Sales']         # Single brackets: returns Series

# Create and fit model
model = LinearRegression()
model.fit(X, y)         # <-- This runs the normal equation!

# Access the learned parameters
print(f"Intercept (beta_0): {model.intercept_}")
print(f"Slope (beta_1): {model.coef_[0]}")

# Make predictions
prediction = model.predict([[150]])  # Predict sales for TV budget = $150k
```

Listing 1: Simple Linear Regression with sklearn

Warning

Why Double Brackets for X?

sklearn expects X to be a 2D array (matrix), even with one predictor:

- `df['TV']` returns a **Series** with shape `(n,)` — **Error!**
- `df[['TV']]` returns a **DataFrame** with shape `(n, 1)` — **Works!**

4 Multiple Linear Regression (MLR)

4.1 From One to Many Predictors

Real predictions often require multiple inputs:

- Height prediction: weight, biological sex, parents' heights
- Sales prediction: TV budget, radio budget, newspaper budget
- House price: square footage, bedrooms, location, age

Definition:

Multiple Linear Regression

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon$$

Interpretation of β_j: The change in Y for a 1-unit change in X_j, **holding all other predictors constant**.

4.2 Matrix Notation

With many predictors, writing out $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ...$ gets tedious. Matrix notation is cleaner:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

- $\mathbf{Y}$: Response vector $(n \times 1)$
- $\mathbf{X}$: Design matrix $(n \times (p+1))$
- $\boldsymbol{\beta}$: Coefficient vector $((p+1) \times 1)$

4.2.1 The Design Matrix

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$

Warning

Why the Column of 1s?

The first column (all 1s) is a mathematical trick to include the intercept β_0 in the matrix multiplication:

$$\begin{pmatrix} 1 & x_1 & x_2 \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} = \beta_0 \cdot 1 + \beta_1 x_1 + \beta_2 x_2$$

Without the 1s column, we'd need to handle β_0 separately.

4.3 The Normal Equation for Multiple Regression

The same logic (set derivatives to zero) gives us the matrix form:

The Normal Equation (Closed-Form Solution)

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

This single formula gives you all optimal coefficients at once!

Key Information

This Only Works for Linear Regression

The closed-form solution is special. For most other models (logistic regression, neural networks, decision trees), no such formula exists. You need iterative methods like gradient descent.

That's why we **love** linear regression—it's mathematically elegant.

4.4 Implementation with Multiple Predictors

```python
# Multiple predictors
X = df[['TV', 'Radio', 'Newspaper']]  # Shape: (n, 3)
y = df['Sales']

model = LinearRegression()
model.fit(X, y)

print(f"Intercept: {model.intercept_}")
print(f"Coefficients: {model.coef_}")
# Output might be: [0.046, 0.189, -0.001]
# Interpretation: Radio has strongest effect, Newspaper almost none
```

Listing 2: Multiple Linear Regression

5 Interpreting Coefficients

5.1 What Does a Coefficient Mean?

Definition:

Coefficient Interpretation In the model $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ...$:
β_j represents the expected change in Y for a **one-unit increase in X_j, holding all other predictors constant**.

Example:

Advertising Example Model: Sales $= 2.94 + 0.046 \cdot \text{TV} + 0.189 \cdot \text{Radio} - 0.001 \cdot \text{Newspaper}$
Interpretations:

- **TV (0.046)**: Each additional \$1,000 in TV budget increases sales by 46 units (holding Radio and Newspaper constant)
- **Radio (0.189)**: Each additional \$1,000 in Radio budget increases sales by 189 units—much larger effect!
- **Newspaper (-0.001)**: Almost zero effect; possibly not worth including

5.2 Feature Importance

To compare which predictors matter most, look at **coefficient magnitudes**:

Warning

Danger: Different Scales!
If TV is in thousands (\$0–300k) and Radio is in hundreds (\$0–50k), comparing raw coefficients is misleading!
A "small" coefficient on a large-scale variable might have more total effect than a "large" coefficient on a small-scale variable.
Solution: **Scale your features** before comparing.

5.3 Coefficient Interpretation Summary

- $\beta = 0$: Predictor has no effect (can be removed)
- $\beta > 0$: Positive relationship (increase $X \rightarrow$ increase Y)
- $\beta < 0$: Negative relationship (increase $X \rightarrow$ decrease Y)
- $|\beta|$ **large**: Strong effect (but check the scale!)

6 Feature Scaling

6.1 Why Scale?

Different predictors often have different units and ranges:

- Age: 0–100
- Income: \$0–\$10,000,000
- Height: 150–200 cm

Without scaling, coefficients can't be directly compared.

6.2 Standardization (Z-Score)

Definition:

Standardization Transform each feature to have mean 0 and standard deviation 1:

$$z = \frac{x - \mu}{\sigma}$$

After standardization, **all features are on the same scale**. A coefficient of 0.5 means "a 1-standard-deviation increase in X produces a 0.5-unit change in Y."

6.3 Min-Max Scaling (Normalization)

Definition:

Min-Max Scaling Transform each feature to the range $[0, 1]$:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

6.4 Does Scaling Affect Predictions?

Key Information

For Linear Regression: No!
Scaling predictors changes the coefficient values but **not the predictions**. If you scale X by 2, the coefficient gets divided by 2—the final prediction is identical.
Why scale then?
1. **Interpretation**: Compare feature importance fairly
2. **Numerical stability**: Prevents very large/small numbers in computations
3. **Other algorithms**: kNN, neural networks **require** scaling

7 Collinearity

7.1 What is Collinearity?

Definition:

Collinearity **Collinearity** (or multicollinearity) occurs when two or more predictors are highly correlated with each other.

Example: Credit card "Limit" and "Rating" are nearly identical—one is probably computed from the other.

7.2 Why is Collinearity a Problem?

Warning

Collinearity Confuses Coefficient Interpretation

If X_1 and X_2 are perfectly correlated:

- When X_1 increases, X_2 also increases
- The model can't tell which one is "causing" the change in Y
- Coefficients become unstable and hard to interpret

Mathematical issue: $(X^T X)^{-1}$ becomes unstable when columns of X are nearly identical.

7.3 Detecting Collinearity

1. **Correlation matrix**: Look for high correlations ($|r| > 0.8$) between predictors
2. **Scatter plot matrix**: Visualize relationships between all pairs
3. **VIF (Variance Inflation Factor)**: Formal measure (covered later)

7.4 What Collinearity Doesn't Break

Key Information

Collinearity is OK for prediction!

If you only care about **predictions** (not interpretation), collinearity doesn't hurt MSE. The predictions will still be accurate.

It only matters when you want to **interpret** which variables are important.

7.5 Handling Collinearity

1. **Remove one of the correlated variables**
2. **Combine them** into a single feature
3. **Use regularization** (Ridge regression—covered later)
4. **Accept it** if you only care about prediction

8 Categorical Predictors

8.1 The Problem with Categories

Linear regression requires **numbers**. But what about:

- Gender: Male/Female
- Color: Red/Blue/Green
- Education: High School/Bachelor's/Master's/PhD

You can't multiply $\beta \times$ "Male"!

8.2 Dummy Variables (One-Hot Encoding)

Definition:

Dummy Variables Convert each category into a **binary (0/1) column**:

Original	IsMale	IsFemale	IsOther
Male	1	0	0
Female	0	1	0
Female	0	1	0
Other	0	0	1

8.3 The Dummy Variable Trap

Warning

Drop One Category!

If you include **all** dummy columns, they sum to 1 (perfect collinearity with the intercept).

Solution: Drop one category (the "reference" or "baseline").

With only IsMale and IsFemale:

- If IsMale = 0 and IsFemale = 0, we know it's "Other"
- The "Other" effect is absorbed into the intercept

8.4 Implementation

```python
import pandas as pd

# Create dummy variables
df_dummies = pd.get_dummies(df, columns=['Gender'], drop_first=True)
# drop_first=True avoids the dummy variable trap

# Now 'Gender_Male', 'Gender_Other' are included (Female is baseline)
X = df_dummies[['Income', 'Age', 'Gender_Male', 'Gender_Other']]
```

Listing 3: Handling Categorical Variables

9 Key Takeaways

Key Summary

Summary of Lecture 05: Linear Regression

The Big Picture

- Linear regression follows the universal ML recipe: Model → Loss → Minimize
- Unlike kNN, it gives **interpretable coefficients**

Simple Linear Regression

- Model: $\hat{Y} = \beta_0 + \beta_1 X$
- Loss: MSE $= \frac{1}{n}\sum(y_i - \hat{y}_i)^2$
- Solution: Set derivatives to zero → Normal equation

Multiple Linear Regression

- Model: $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta}$
- Solution: $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$
- The column of 1s in X handles the intercept

Interpretation

- β_j: Change in Y for 1-unit change in X_j, holding others constant
- Scale features before comparing coefficient magnitudes

Practical Issues

- **Collinearity**: Confuses interpretation (not predictions)
- **Categorical variables**: Use dummy encoding, drop one category
- **Scaling**: Doesn't change predictions, but helps interpretation

9.1 Looking Ahead

Next lectures will cover:

- **Polynomial regression**: What if the relationship isn't linear?
- **Model selection**: How to choose which predictors to include?
- **Regularization**: Ridge and Lasso regression
- **Assumptions of linear regression**: What makes it valid?