Artificial Intelligence

Lecture 7: Introduction to Machine Learning

Prof. Antonio Khalil Moretti

Week 8

SCIS 432
Spelman College

A Different Approach: Learning from Data

Classical AI vs. Machine Learning

Classical AI

Explicit programming:

- Write rules by hand
- Encode expert knowledge
- Design algorithms

Examples:

- Search algorithms (A*)
- Constraint satisfaction
- Game playing (Minimax)

Challenge:
Hard to write rules for complex patterns (e.g., recognizing faces, understanding speech)

Machine Learning

Learning from data:

- Show examples
- Algorithm finds patterns
- Generalizes to new cases

Examples:

- Image recognition
- Speech recognition
- Recommendation systems

Advantage:
Discovers patterns too complex or subtle for humans to articulate

Historical Perspective: ML as Statistics

Machine learning has deep roots in statistics and mathematical optimization

1763	1805	1950s	1980s	2012
Bayes' Theorem	Least Squares	Perceptron	Backpropagation	Deep Learning Era
Foundation for probabilistic inference	Regression	Learning algorithm	Training deep neural networks	AlexNet wins ImageNet

- **1763–1805:** Mathematical foundations (probability, optimization)
- **1950s–1980s:** Early neural networks and learning algorithms
- **1990s–2000s:** Statistical learning theory, SVMs, ensemble methods
- **2012–present:** Deep learning revolution enabled by big data + GPUs

Two Main Types of Supervised Learning

Classification

Predict a category or class

Output: Discrete label

Examples:

- Is this email spam? (Yes/No)
- What digit is this? (0–9)
- Is this tumor malignant? (Yes/No)
- What animal is in this image? (cat/dog/bird/...)

Key property:
Output belongs to a finite set of categories

Regression

Predict a continuous value

Output: Real number

Examples:

- What will the temperature be?
- How much will this house sell for?
- What will stock price be tomorrow?
- How many units will we sell next month?

Key property:
Output is a number on a continuous scale

Today's focus: Linear Regression

Linear Regression: The Simplest Model

Example: Predicting apartment rent from square footage

The Problem

We have data from 10 apartments:

Sq Ft	Rent ($)
620	1,150
750	1,380
850	1,610
950	1,710
1,050	1,920
1,150	2,000
1,280	2,250
1,400	2,290
1,520	2,600
1,620	2,710

The Goal

Predict rent for a *new* apartment
(e.g., 900 sq ft or 1,300 sq ft)

The Approach

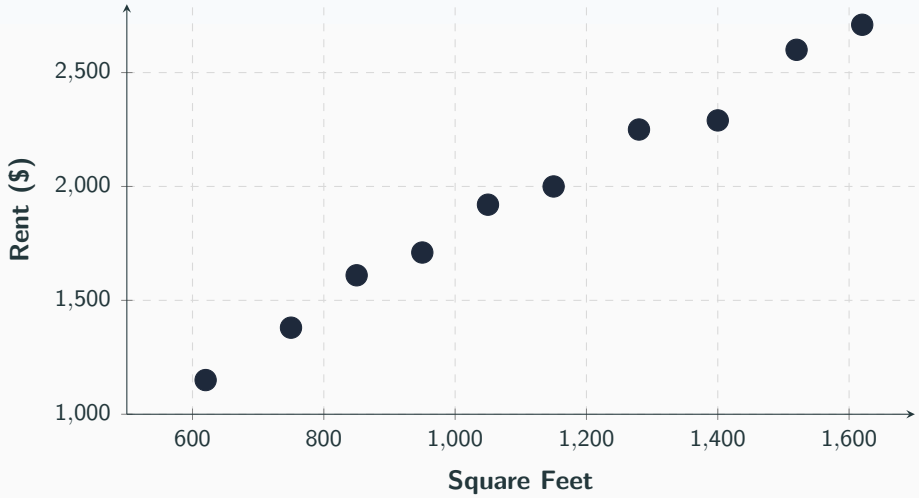
Step 1: Plot the data points

Step 2: Fit a line through the points

Step 3: Use the line to predict new values

Key insight:
If the relationship is approximately linear, a line captures the pattern and allows us to interpolate (and extrapolate)

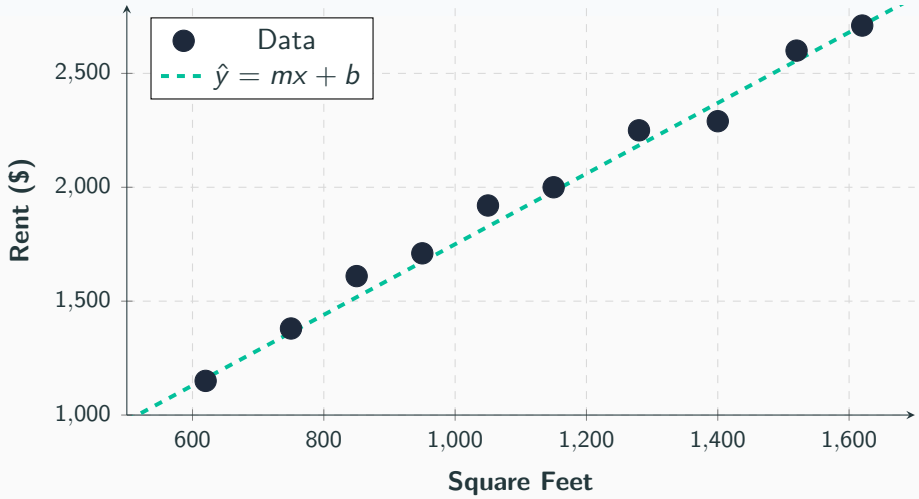
Visualizing the Data: Scatterplot



Each point represents one apartment: (square footage, monthly rent)

Notice the data doesn't fall perfectly on a line — this is realistic!

Fitting a Line Through the Points



The line doesn't pass through all points exactly — it finds the best overall fit

The Learning Problem

What can we change? What is fixed?

Given (Fixed)

Data points (x, y)**:**

- (600, 1200)
- (800, 1500)
- (1000, 1800)
- (1200, 2100)
- (1500, 2400)

We **cannot** change these!
They are observations from the real world.

Choose (Parameters)

Model parameters:

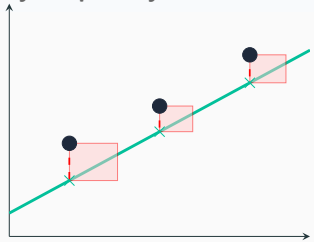
- Slope m
- Intercept b

We **must choose** these to make the line fit the data as well as possible.

Question:
How do we pick the *best* values of m and b?

Measuring How Well a Line Fits

We need a way to quantify "error" or "loss"



• Actual y
× Predicted $\hat{y}$
— Error

Sum of Squared Errors (SSE)

For each data point:

- Actual value: y_i
- Predicted value: $\hat{y}_i = mx_i + b$
- Error: $e_i = y_i - \hat{y}_i$

Total error:

$$\text{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Why square?

- Positive and negative errors don't cancel
- Penalizes large errors more
- Mathematically convenient

Calculus Review: Derivatives

The derivative measures the rate of change

Geometric Interpretation

The derivative $\frac{df}{dx}$ is the **slope of the tangent line** to $f(x)$ at a point.

Key fact:
When $\frac{df}{dx} = 0$, the function has:

- A local maximum
- A local minimum
- Or a saddle point

For **convex** functions:
Setting derivative to zero gives the **global minimum**

Example: $f(x) = x^2$

$$f(x) = x^2$$

$$\frac{df}{dx} = 2x$$

Set derivative to zero:

$$2x = 0 \quad \Rightarrow \quad x = 0$$

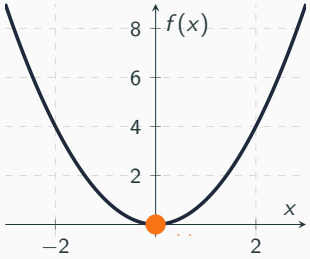
At $x = 0$:

- $f(0) = 0$
- This is the **minimum** of the parabola

We use this same principle to minimize SSE!

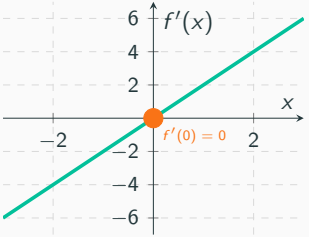
Visualizing $f(x) = x^2$ and Its Derivative

Function: $f(x) = x^2$



The parabola has a minimum at $x = 0$ where $f(0) = 0$

Derivative: $f'(x) = 2x$

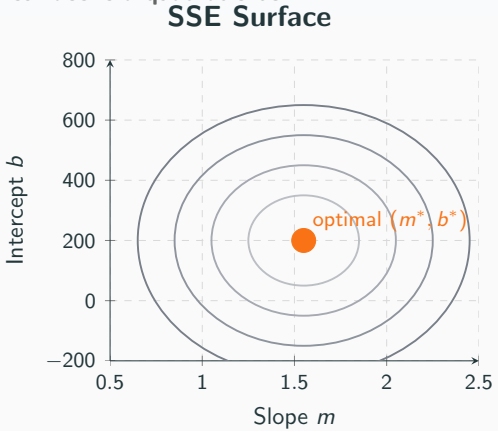


The derivative crosses zero at $x = 0$ — this is where $f(x)$ has its minimum!

Key observation: Where the derivative equals zero, the original function has a critical point (minimum, maximum, or saddle point)

SSE as a Function of Parameters m and b

The error surface is a quadratic bowl



Each contour line represents points with the same SSE value

Understanding the Surface

The Sum of Squared Errors is a **quadratic function** of both m and b:

$$\text{SSE}(m, b) = \sum_i (y_i - mx_i - b)^2$$

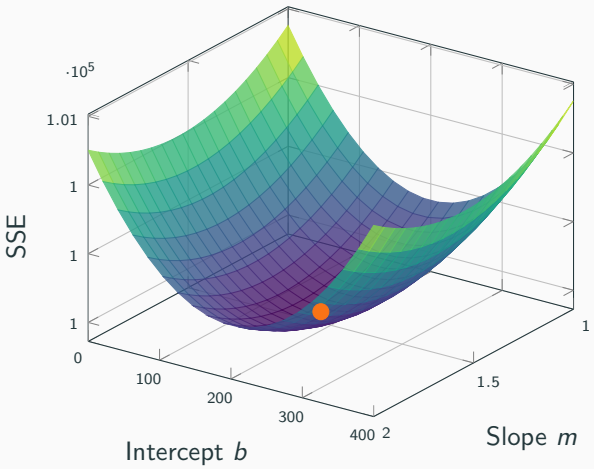
Properties:

- Convex (bowl-shaped)
- Has a unique global minimum
- Setting both partial derivatives to zero finds the minimum

Optimal parameters:
The point (m^*, b^*) at the bottom of the bowl minimizes SSE

SSE as a Function of Parameters m and b

3D SSE Surface



The bowl shape shows SSE increases as we move away from the optimal (m^*, b^*)

Understanding the Surface

The Sum of Squared Errors is a **quadratic function** of both m and b:

$$\text{SSE}(m, b) = \sum_i (y_i - mx_i - b)^2$$

Properties:

- Convex (bowl-shaped)
- Has a unique global minimum
- Setting both partial derivatives to zero finds the minimum

The red point at the bottom is where:

$$\frac{\partial \text{SSE}}{\partial m} = 0, \quad \frac{\partial \text{SSE}}{\partial b} = 0$$

Finding the Best Line: Minimize SSE

Step 1: Write out the SSE formula

$$\text{SSE}(m, b) = \sum_{i=1}^{n} (y_i - (mx_i + b))^2$$

This is a function of two variables: m and b
To minimize, we take **partial derivatives** with respect to each parameter and set them to zero.

Step 2: Take partial derivatives

$$\frac{\partial(\text{SSE})}{\partial m} = 0$$

$$\frac{\partial(\text{SSE})}{\partial b} = 0$$

These give us a **system of two equations** with two unknowns (m and b).
Solving this system gives us the optimal parameters!

Derivation: Partial Derivative w.r.t. b

Start with SSE:

$$\text{SSE} = \sum_{i=1}^{n} (y_i - mx_i - b)^2$$

Take partial derivative with respect to b using chain rule:

$$\frac{\partial(\text{SSE})}{\partial b} = \sum_{i=1}^{n} 2(y_i - mx_i - b)(-1)$$

Simplify:

$$\frac{\partial(\text{SSE})}{\partial b} = -2\sum_{i=1}^{n} (y_i - mx_i - b)$$

Set equal to zero:

$$\sum_{i=1}^{n} (y_i - mx_i - b) = 0$$

Derivation: Partial Derivative w.r.t. m

Same SSE:

$$\text{SSE} = \sum_{i=1}^{n} (y_i - mx_i - b)^2$$

Take partial derivative with respect to m using chain rule:

$$\frac{\partial(\text{SSE})}{\partial m} = \sum_{i=1}^{n} 2(y_i - mx_i - b)(-x_i)$$

Simplify:

$$\frac{\partial(\text{SSE})}{\partial m} = -2 \sum_{i=1}^{n} x_i(y_i - mx_i - b)$$

Set equal to zero:

$$\sum_{i=1}^{n} x_i(y_i - mx_i - b) = 0$$

Solving the System of Equations

We now have two equations (called the **normal equations**):

$$\text{Equation 1:} \quad \sum_{i=1}^{n}(y_i - mx_i - b) = 0$$

$$\text{Equation 2:} \quad \sum_{i=1}^{n} x_i(y_i - mx_i - b) = 0$$

Expand Equation 1:

$$\sum_{i=1}^{n} y_i - m\sum_{i=1}^{n} x_i - nb = 0$$

Solve for b:

$$b = \bar{y} - m\bar{x}$$

Solving for the Slope m

Substitute $b = \bar{y} - m\bar{x}$ into Equation 2:

$$\sum_{i=1}^{n} x_i \left(y_i - mx_i - (\bar{y} - m\bar{x})\right) = 0$$

Expand and rearrange:

$$\sum_{i=1}^{n} x_i y_i - m\sum_{i=1}^{n} x_i^2 - \bar{y}\sum_{i=1}^{n} x_i + m\bar{x}\sum_{i=1}^{n} x_i = 0$$

$$\sum_{i=1}^{n} x_i y_i - n\bar{y}\bar{x} = m\left(\sum_{i=1}^{n} x_i^2 - n\bar{x}^2\right)$$

Therefore:

$$m = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2}$$

Alternative Form: Covariance and Variance

The formula for m can be rewritten more compactly:

$$m = \frac{\text{Cov}(x, y)}{\text{Var}(x)}$$

where:

$$\text{Cov}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) = \frac{1}{n}\sum_{i=1}^{n} x_i y_i - \bar{x}\bar{y}$$

$$\text{Var}(x) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2$$

Interpretation:
The slope is the ratio of how x and y co-vary to how much x varies.

Covariance and Correlation: Definitions

Covariance measures the joint variability of two variables:

$$\text{Cov}(x, y) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

- $\text{Cov}(x, y) > 0$: x and y tend to move **together**
- $\text{Cov}(x, y) < 0$: x and y tend to move **oppositely**
- $\text{Cov}(x, y) = 0$: no linear relationship

Correlation standardizes covariance to $[-1, 1]$:

$$r = \frac{\text{Cov}(x, y)}{\sqrt{\text{Var}(x)} \cdot \sqrt{\text{Var}(y)}} = \frac{\text{Cov}(x, y)}{\sigma_x \, \sigma_y}$$

- $r = \pm 1$: perfect linear relationship $\quad$ $r = 0$: no linear relationship
- Unlike covariance, r is **unitless** and scale-invariant

Covariance and Correlation: Example

Consider $n = 4$ observations of study hours x and exam score y:

i	x_i	y_i	$x_i - \bar{x}$	$y_i - \bar{y}$	$(x_i - \bar{x})(y_i - \bar{y})$
1	1	50	-2	-20	40
2	2	60	-1	-10	10
3	4	80	$+1$	$+10$	10
4	5	90	$+2$	$+20$	40
$\bar{x} = 3$		$\bar{y} = 70$			$\sum = 100$

$$\text{Cov}(x, y) = \frac{100}{4} = 25 \quad > 0 \Rightarrow \text{positive relationship}$$

$$\text{Var}(x) = \frac{1}{4}\left[(-2)^2 + (-1)^2 + 1^2 + 2^2\right] = \frac{10}{4} = 2.5$$

$$\text{Var}(y) = \frac{1}{4}\left[400 + 100 + 100 + 400\right] = 250$$

$$r = \frac{25}{\sqrt{2.5} \cdot \sqrt{250}} = \frac{25}{25} = \mathbf{1.0} \Rightarrow \text{perfect linear fit}$$

The Closed-Form Solution

We now have **explicit formulas** for m and b:

$$\text{Slope:} \quad m = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2} = \frac{\text{Cov}(x, y)}{\text{Var}(x)}$$

$$\text{Intercept:} \quad b = \bar{y} - m\bar{x}$$

Properties of this solution:

- ✓ Minimizes sum of squared errors
- ✓ Exact (not approximate)
- ✓ Computable directly from data
- ✓ The fitted line always passes through $(\bar{x}, \bar{y})$
- ✓ Unique solution (for non-degenerate data)

This is called **Ordinary Least Squares (OLS) regression**, *the foundation of statistical learning.*

Why OLS? The Gauss–Markov Theorem

A theoretical guarantee for linear regression

The Theorem

Under the following assumptions, OLS produces the **Best Linear Unbiased Estimator (BLUE)**:

Gauss–Markov Assumptions:

1. **Linearity:** $y_i = mx_i + b + \varepsilon_i$
2. **Strict exogeneity:** $\mathbb{E}[\varepsilon_i] = 0$
   (errors have zero mean)
3. **Homoscedasticity:** $\mathrm{Var}(\varepsilon_i) = \sigma^2$
   (equal variance across all points)
4. **No autocorrelation:** $\mathrm{Cov}(\varepsilon_i, \varepsilon_j) = 0$ for $i \neq j$
   (errors are uncorrelated with each other)

Note: normality of errors is **not** *required!*

What "BLUE" means

Among all **linear unbiased** estimators, OLS has the **smallest variance**.

Best	Minimum variance (most precise)
Linear	Estimator is a linear function of the data $y_1, \ldots, y_n$
Unbiased	On average, recovers the true m and b
Estimator	Uses only the observed data

Practical implication:
If the assumptions hold, no other method can beat OLS at estimating the true line — it is the theoretically optimal choice.
When assumptions are violated (e.g., heteroscedasticity), other estimators may outperform OLS.

From One Feature to Many

Why we need a more compact notation

Simple linear regression

One feature, one equation:

$$\hat{y} = mx + b$$

For 4 data points, our predictions are:

$$\hat{y}_1 = mx_1 + b$$
$$\hat{y}_2 = mx_2 + b$$
$$\hat{y}_3 = mx_3 + b$$
$$\hat{y}_4 = mx_4 + b$$

Two unknowns: m and b

Multiple linear regression

Three features (e.g., sq ft, bedrooms, distance to transit):

$$\hat{y} = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_0$$

For 4 data points:

$$\hat{y}_1 = \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \beta_3 x_{13}$$
$$\hat{y}_2 = \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \beta_3 x_{23}$$
$$\hat{y}_3 = \beta_0 + \beta_1 x_{31} + \beta_2 x_{32} + \beta_3 x_{33}$$
$$\hat{y}_4 = \beta_0 + \beta_1 x_{41} + \beta_2 x_{42} + \beta_3 x_{43}$$

Four unknowns: $\beta_0, \beta_1, \beta_2, \beta_3$

Problem: Writing out every equation becomes unmanageable.

Solution: Package all the data and parameters into **matrices and vectors**.

Matrix Multiplication

The rule: row $\times$ column

To multiply $\mathbf{A}$ $(m \times k)$ by $\mathbf{B}$ $(k \times n)$:

- Inner dimensions must match: $k = k$
- Result has shape $m \times n$
- Entry (i, j) = dot product of row i of $\mathbf{A}$ with column j of $\mathbf{B}$

$$(\mathbf{AB})_{ij} = \sum_{\ell=1}^{k} a_{i\ell}\, b_{\ell j}$$

Example:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \end{pmatrix} = \begin{pmatrix} 1{\cdot}5 + 2{\cdot}6 \\ 3{\cdot}5 + 4{\cdot}6 \end{pmatrix} = \begin{pmatrix} 17 \\ 39 \end{pmatrix}$$

Connecting to regression

Stack all data points into a matrix $\mathbf{X}$ (with a column of 1s for the intercept) and parameters into $\boldsymbol{\beta}$:

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} b \\ m \end{pmatrix}$$

Then the matrix product gives *all* predictions at once:

$$\mathbf{X}\boldsymbol{\beta} = \begin{pmatrix} b + mx_1 \\ b + mx_2 \\ b + mx_3 \\ b + mx_4 \end{pmatrix} = \hat{\mathbf{y}}$$

One compact expression replaces all n prediction equations!

Key properties: $\mathbf{AB} \neq \mathbf{BA}$ in general (order matters), but $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$ (associative).

Matrices and Vectors

Key objects

A **matrix A** is a rectangular array of numbers with m rows and n columns:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \quad (3 \times 2)$$

A **column vector** is a matrix with one column:

$$\mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \quad (3 \times 1)$$

A **row vector** is a matrix with one row:

$$\mathbf{u} = \begin{pmatrix} u_1 & u_2 & u_3 \end{pmatrix} \quad (1 \times 3)$$

The transpose

The **transpose** $\mathbf{A}^T$ flips a matrix over its diagonal, rows become columns:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}_{3\times 2} \quad \Rightarrow \quad \mathbf{A}^T = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}_{2\times 3}$$

Rules for transpose:

- $(\mathbf{A}^T)^T = \mathbf{A}$
- $(c\mathbf{A})^T = c\mathbf{A}^T$
- $(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$
- $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$ *(order reverses!)*

The last rule is critical, we will use it when deriving the normal equation.

Why Does $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$?

Let $\mathbf{A}$ be $m \times k$ and $\mathbf{B}$ be $k \times n$, so $\mathbf{AB}$ is $m \times n$.

We want to show that $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$.

Strategy: show both sides have the same shape, then show every (i, j) entry matches.

Check the shapes first

Matrix	Shape	
$\mathbf{AB}$	$m \times n$	
$(\mathbf{AB})^T$	$n \times m$	*(transpose flips)*
$\mathbf{B}^T$	$n \times k$	
$\mathbf{A}^T$	$k \times m$	
$\mathbf{B}^T\mathbf{A}^T$	$n \times m$	✓ shapes match

Proof: compare the (i, j) entries The (i, j) entry of $(\mathbf{AB})^T$ is the (j, i) entry of $\mathbf{AB}$:

$$\left[(\mathbf{AB})^T\right]_{ij} = [\mathbf{AB}]_{ji} = \sum_{\ell=1}^{k} a_{j\ell}\, b_{\ell i}$$

Now compute the (i, j) entry of $\mathbf{B}^T\mathbf{A}^T$. Recall $[\mathbf{B}^T]_{i\ell} = b_{\ell i}$ and $[\mathbf{A}^T]_{\ell j} = a_{j\ell}$, so:

$$\left[\mathbf{B}^T\mathbf{A}^T\right]_{ij} = \sum_{\ell=1}^{k} [\mathbf{B}^T]_{i\ell}\, [\mathbf{A}^T]_{\ell j} = \sum_{\ell=1}^{k} b_{\ell i}\, a_{j\ell}$$

Addition is commutative, $a_{j\ell}\, b_{\ell i} = b_{\ell i}\, a_{j\ell}$, so:

$$\sum_{\ell=1}^{k} a_{j\ell}\, b_{\ell i} = \sum_{\ell=1}^{k} b_{\ell i}\, a_{j\ell}$$

Both sides are equal for every (i, j). ■

Insight: transposing "un-does" the row×column flip, so the product order must reverse to compensate.

Matrix Formulation (Advanced)

For multiple features, we use linear algebra

Setup:

- n data points
- d features per point
- $\mathbf{X} = n \times d$ matrix of inputs
- $\mathbf{y} = n \times 1$ vector of outputs
- $\boldsymbol{\beta} = d \times 1$ vector of parameters

Model:

$$\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}$$

Loss function:

$$\text{SSE}(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2$$

$$= (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$

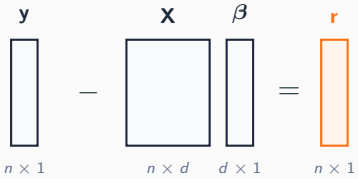
Take derivative and set to zero:

$$\frac{\partial(\text{SSE})}{\partial\boldsymbol{\beta}} = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = \mathbf{0}$$

Note: Matrix notation provides a compact way to represent and solve systems of linear equations with multiple variables.

Unpacking $(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$

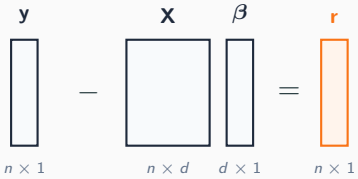
Let $\mathbf{r} = \mathbf{y} - \mathbf{X}\boldsymbol{\beta}$ (the residual vector). Then SSE $= \mathbf{r}^T\mathbf{r}$.



Each entry $r_i = y_i - \boldsymbol{\beta}^T\mathbf{x}_i$ is the residual for point i

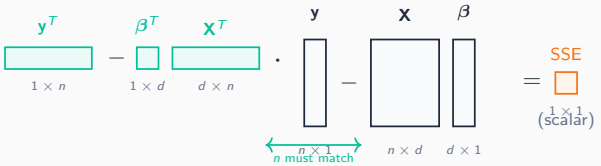
Unpacking $(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$

Let $\mathbf{r} = \mathbf{y} - \mathbf{X}\boldsymbol{\beta}$ (the residual vector). Then SSE $= \mathbf{r}^T\mathbf{r}$.



Each entry $r_i = y_i - \boldsymbol{\beta}^T\mathbf{x}_i$ is the residual for point i

$$\text{SSE} = \mathbf{r}^T\mathbf{r} = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$$



The transpose of a product reverses order: $(\mathbf{X}\boldsymbol{\beta})^T = \boldsymbol{\beta}^T\mathbf{X}^T$ (the rule we just proved).

The inner dimensions ($n \times n$) contract to leave a 1×1 scalar: the sum of squared residuals.

The Normal Equation

From:

$$-2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = \mathbf{0}$$

Expand:

$$\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{0}$$

Rearrange:

$$\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}$$

Solution (if $\mathbf{X}^T\mathbf{X}$ is invertible):

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

This is the **normal equation**: the closed-form solution for multivariate linear regression.

The matrix inverse $(\mathbf{X}^T\mathbf{X})^{-1}$ effectively solves the system of equations derived from setting all partial derivatives to zero.

Implementing the Normal Equation in Python

Direct inverse (when $\mathbf{X}^T\mathbf{X}$ is invertible):

```
import numpy as np
# X: (n x d) design matrix,  y:  (n x 1) target vector
XtX = X.T @ X          # (d x d)
Xty = X.T @ y          # (d x 1)
beta = np.linalg.inv(XtX) @ Xty   # exact inverse
```

Recommended: pseudoinverse (robust, always works):

```
beta = np.linalg.pinv(X) @ y    # Moore--Penrose pseudoinverse
```

When does `inv` fail?

$\mathbf{X}^T\mathbf{X}$ is **not invertible** when:

- $\det(\mathbf{X}^T\mathbf{X}) = 0$ (singular matrix)
- Features are linearly dependent (multicollinearity)
- $n < d$: fewer samples than features

Near-zero determinant signals numerical instability

Why `pinv` is safer

Moore–Penrose pseudoinverse $\mathbf{X}^+$ generalises inverse:

- Uses SVD: $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$
- Inverts only *non-zero* singular values
- Returns the **minimum-norm** least-squares solution
- Works even when $\mathbf{X}^T\mathbf{X}$ is singular

Summary: What We Learned

Conceptual:

- ML learns patterns from data
- Regression predicts continuous values
- Linear regression fits a line to data
- We minimize squared prediction errors
- Calculus finds the optimal parameters

Mathematical:

- Model: $y = mx + b$
- Loss: SSE $= \sum(y_i - \hat{y}_i)^2$
- Solution via calculus:
  - Set $\frac{\partial(\text{SSE})}{\partial m} = 0$
  - Set $\frac{\partial(\text{SSE})}{\partial b} = 0$
  - Solve system of equations
- Matrix form: $\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

Practical:

- Real-world applications:
  - Price prediction (housing, stocks)
  - Demand forecasting
  - Risk assessment
  - Scientific modeling
  - Trend analysis
- Implementation:
  - Can compute directly from data
  - No iteration needed
  - Built into libraries (scikit-learn)

Next Time:

- Multiple linear regression
- Gradient descent
- Overfitting and regularization
- Evaluation metrics

Questions?

Next week:

Advanced regression techniques and gradient descent

Recommended reading:

- Murphy, *Probabilistic Machine Learning: An Introduction*, Chapter 11
- Bishop, *Pattern Recognition and Machine Learning*, Chapter 3
- Hastie et al., *The Elements of Statistical Learning*, Chapter 3