Linear Regression I

Linear Regression Hypothesis

1. Detailed Conceptual Overview

Linear Regression is one of the most fundamental algorithms in Machine Learning. It assumes that the relationship between the input features (**independent variables**) and the output (**dependent variable**) can be approximated using a **linear function**.

Core Idea:

- Predict y as a **linear combination of input features x**.
- Example: Predict house price using features like size, age, and location score.

General Hypothesis Form:

- For one feature (simple regression):

  $h_\theta(x) = \theta_0 + \theta_1 x$

  Here, θ_0 = intercept (bias), θ_1 = slope (weight).

- For multiple features (multivariate regression):

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d$$

  Compact form (vectorized):

  $h_\theta(x) = \theta^T x$

  where θ = parameter vector, x = feature vector (with a “1” appended for bias).

Interpretation:

- Each coefficient θ_j represents how much the target y changes when feature x_j increases by 1, while keeping other features constant.
- The intercept θ_0 represents the baseline value of y when all features are zero.

Examples:

1. **Simple Regression (1D):** Predicting salary based on years of experience.
   - Line: Salary = Base + (Rate × Years).
2. **Multiple Regression (2D+):** Predicting house price from size and number of rooms.
   - Plane in 3D space: Price = Intercept + w_1·Size + w_2·Rooms.

The linear regression hypothesis forms the **starting point for many ML models** (generalized linear models, logistic regression, neural networks, all extend from here).

---

2. Key Takeaways

- Linear Regression assumes a **linear relationship** between inputs and outputs.
- Hypothesis is a **weighted sum of input features plus an intercept**.
- **Simple Regression:** One input → a line.
- **Multiple Regression:** Many inputs → a hyperplane in higher dimensions.
- Coefficients (θ) capture the **influence of each feature**.
- Baseline intercept (θ_0) shifts the line/plane vertically.
- Forms the **foundation for more advanced ML models**.

Linear Regression Illustration (1D, 2D, Higher Dimensions)

---

1. Detailed Conceptual Overview

To better understand **linear regression**, it helps to **visualize how the hypothesis looks in different dimensions**:

---

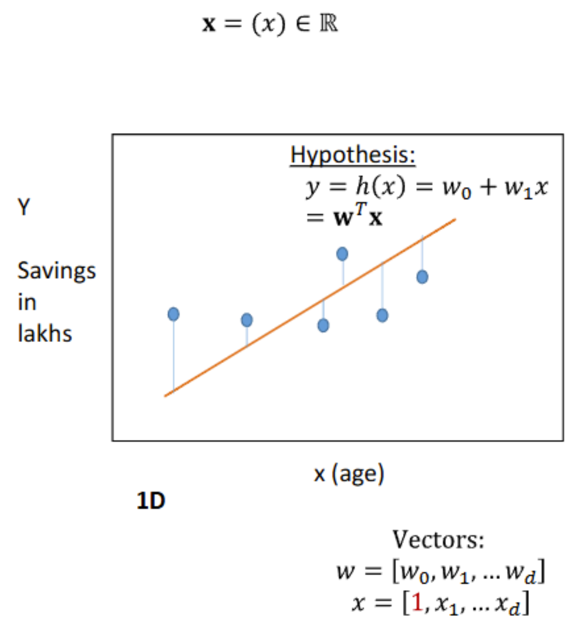
a) 1D Case (Simple Linear Regression):

- One input variable x.
- Hypothesis:

  $$h_\theta(x) = \theta_0 + \theta_1 x$$

- Graph: A **straight line** in 2D space (x-axis = input, y-axis = output).
- Example: Predicting salary based on years of experience.

👉 Interpretation: Increasing x by 1 increases prediction by slope (θ_1).



---

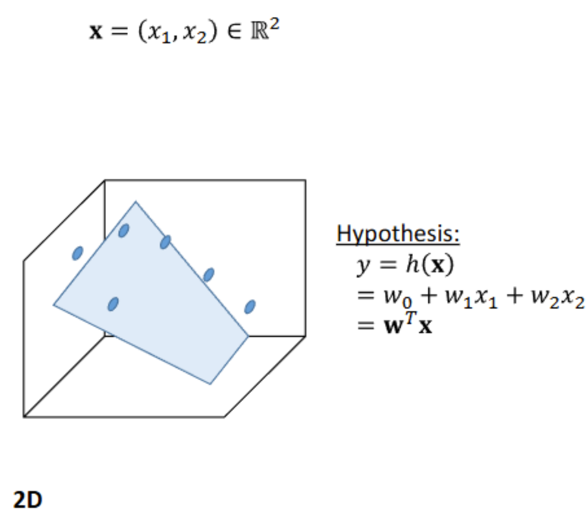
b) 2D Case (Two Input Features):

- Two input variables (x_1, x_2).
- Hypothesis:

  $$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

- Graph: A **plane** in 3D space.
- Example: Predicting house price based on size (x_1) and number of rooms (x_2).

👉 Interpretation: Each coefficient adjusts the slope of the plane along that feature axis.



---

c) Higher Dimensions (Multivariate Regression):

- Many features $(x_1, x_2, \dots, x_d)$.
- Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d$$

- Graph: A **hyperplane** in d+1 dimensional space.
- Cannot visualize beyond 3D, but mathematically, the same principle applies.
- Example: Predicting medical costs using features like age, BMI, smoking habits, and number of dependents.

Role of the Intercept (θ_0):

- Shifts the line/plane/hyperplane up or down.
- Ensures predictions aren't forced to pass through the origin (0,0,...,0).
- Example: Even if a house has size = 0, the intercept may represent baseline cost (land value, taxes, etc.).

---

2. Key Takeaways

- **1D Linear Regression:** Line fit between input & output.
- **2D Linear Regression:** Plane fit between two inputs & output.
- **Higher-Dimensional Regression:** Hyperplane fit for many features.
- Visualization becomes harder as dimensions increase, but math is the same.
- Intercept shifts the regression surface vertically.
- Each feature contributes a **slope** (weight), controlling how strongly it affects predictions.

Choosing the Best Coefficients (Error Minimization in Linear Regression)

---

1. Detailed Conceptual Overview

In Linear Regression, our hypothesis is:

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d$

But how do we **find the best coefficients (θ)**?

The idea:

- The model should make predictions **as close as possible to actual outputs**.
- We measure closeness using an **error function (loss function)**.
- The "best" coefficients are those that **minimize this error across the training dataset**.

How it works:

1. Take all training points (x_i, y_i).
2. Compute prediction error:

   $\text{error}_i = y_i - h_\theta(x_i)$
3. Aggregate errors into a **cost function** (commonly **Sum of Squared Errors**):

   $J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2$
4. Find θ that minimizes J(θ).

👉 This is essentially an **optimization problem**.

- For small problems → we can solve analytically (normal equation).
- For large problems → we use iterative optimization (like Gradient Descent).

Why Squared Error?

- Penalizes larger mistakes more heavily.
- Makes the function smooth and differentiable, useful for optimization.
- Leads to a unique solution if features are independent.

---

2. Key Takeaways

- Best coefficients are chosen by **minimizing prediction error**.
- Error is usually measured as **sum of squared differences** between predicted and actual values.
- This leads to a **cost function J(θ)**, which is minimized to estimate θ.
- Two approaches to minimization:

- **Analytical solution** (Normal Equation).
- **Iterative solution** (Gradient Descent).

- Error minimization ensures that the chosen line/plane/hyperplane is the **"best fit"** for the data.

Error Metrics in Regression (L1, L2, Max Error)

---

1. Detailed Conceptual Overview

When fitting a regression model, we need a way to **measure how good the predictions are** compared to actual values. Different **error metrics** capture different aspects of model performance.

---

a) L1 Norm (Absolute Error):

- Definition: Sum of absolute differences between predictions and actuals.

$$L1 = \sum_{i=1}^{n} |y_i - h_\theta(x_i)|$$

- Properties:
  - More **robust to outliers** (does not exaggerate large errors).
  - Leads to the **Median** as the best-fit estimator in statistics.
- Example usage: Lasso regression (adds L1 penalty).

---

b) L2 Norm (Squared Error):

- Definition: Sum of squared differences between predictions and actuals.

$$L2 = \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2$$

- Properties:
  - Punishes **large errors more strongly** (since errors are squared).
  - Leads to the **Mean** as the best-fit estimator.
  - Most common choice for regression (basis of least squares).

- Example usage: Ridge regression (adds L2 penalty).

---

c) Max Error (Infinity Norm):

- Definition: Largest absolute error across all predictions.

$L_\infty = \max_i |y_i - h_\theta(x_i)|$

- Properties:
  - Focuses on the **worst-case error**.
  - Useful when even a single large mistake is unacceptable (e.g., safety-critical systems).
- Example usage: Quality control, robotics.

Comparison:

- L1: Robust to outliers, but may give less stable solutions.
- L2: Sensitive to outliers, but gives smooth, unique solutions.
- Max Error: Ensures no extreme failure, but ignores average performance.

---

2. Key Takeaways

- **Error metrics quantify model performance in regression.**
- **L1 norm** = absolute error → robust to outliers.
- **L2 norm** = squared error → emphasizes large errors, most common.
- **Max error** = worst-case error → ensures safety-critical reliability.
- Choice of metric depends on application (robustness vs sensitivity vs safety).

Sum of Squared Errors (SSE) & In-sample vs Out-of-sample Error

1. Detailed Conceptual Overview

When fitting a regression model, we need a way to evaluate how well the model explains the data. Two key ideas here are **Sum of Squared Errors (SSE)** and the distinction between **In-sample vs Out-of-sample error**.

---

a) Sum of Squared Errors (SSE):

- Definition: Total squared difference between predicted and actual outputs across all training data.

$SSE = \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2$

- Intuition:
  - Measures the **total error** the model makes on the training set.
  - Squaring emphasizes large deviations more strongly.
- Goal: Minimize SSE when choosing coefficients (θ).
- Example: If predictions match actuals perfectly → SSE = 0.

---

b) In-sample Error (Training Error):

- The error the model makes on the **same data it was trained on**.
- Usually **lower**, since the model has already seen this data.
- Risk: Can be misleading if the model overfits (memorizes training data).
- Formula:

$$E_{in} = \frac{1}{n_{train}} \sum_{i=1}^{n_{train}} (y_i - h(x_i))^2$$

---

c) Out-of-sample Error (Test/Generalization Error):

- The error the model makes on **new, unseen data**.
- More realistic measure of how well the model will perform in practice.
- Usually **higher than in-sample error**, since the model hasn't seen the test data.

- Formula:

$$E_{out} = \mathbb{E}_{(x,y)\sim P}\left[(y - h(x))^2\right]$$

Key Idea:

- A good model should have **both low in-sample and low out-of-sample error**.
- If in-sample error is very low but out-of-sample error is very high → **overfitting**.
- If both errors are high → **underfitting**.

---

2. Key Takeaways

- **SSE (Sum of Squared Errors):** Main cost function used in regression. Squaring emphasizes large deviations.
- **In-sample error:** Model error on training data. It can be artificially low due to overfitting.
- **Out-of-sample error:** Model error on unseen data. True test of generalization.
- ML models must balance between the two: low training error **and** low test error.
- Gap between in-sample and out-of-sample error is a strong indicator of **overfitting**.

Methodology of Regression Solutions (Fitting with 1, 2, or More Points – Least Squares Method)

---

1. Detailed Conceptual Overview

Linear Regression is about finding the **best-fitting line (or hyperplane)** that explains the relationship between input(s) and output. The methodology depends on how many data points we have.

---

a) Fitting with 1 Point:

- If we only have **one data point**, infinitely many lines can pass through it.
- The problem is **underdetermined** → we cannot uniquely define a regression line.
- Example: Point (2, 5). Lines like y = 2x + 1 or y = - x + 7 both fit perfectly.

---

b) Fitting with 2 Points:

- If we have exactly **two data points**, there is exactly **one unique straight line** that passes through both.
- Regression is trivial here → line is perfectly determined.
- Example: Points (1, 2) and (3, 6) → slope = (6–2)/(3–1) = 2, so line = y = 2x.

---

c) Fitting with More than 2 Points:

- With more than two points, in general, **no single line will pass through all points**.
- We must find the **best-fitting line** that minimizes errors.
- Solution: **Least Squares Method**.

Least Squares Method (Core Idea):

- Goal: Minimize total squared error between predictions and actual values.

$$J(\theta) = \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2$$

- Why squares?
  - Eliminates negatives (otherwise positive and negative errors cancel).
  - Amplifies large errors → line fits the bulk of data, not just extremes.
- Result: A unique line that minimizes SSE.

Geometric Interpretation:

- Each data point has a vertical distance (error) from the regression line.

- Least Squares finds the line that minimizes the sum of **squared vertical distances**.

---

2. Key Takeaways

- **1 Point:** Cannot define regression line (underdetermined).
- **2 Points:** Unique line fits perfectly.
- **>2 Points:** Use the **Least Squares Method** to find the best fit.
- **Least Squares Principle:** Minimize total squared error between predictions and actuals.
- This methodology generalizes to multiple regression (planes/hyperplanes).
- Core idea: Regression is about finding the line/hyperplane that **balances errors across all points**.

Algorithmic Solution (Matrix Pseudo-inverse Method)

---

1. Detailed Conceptual Overview

Up to now, we've seen regression as finding the **best-fit line/hyperplane** by minimizing the **sum of squared errors (SSE)**. But how do we **actually compute the coefficients** (θ)?

One powerful approach is the **Matrix Pseudo-inverse Method** — also known as the **Normal Equation Solution**.

---

a) Setup:

- Data matrix:

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1d} \\ 1 & x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix}$$

where each row = one data point, first column = 1 (for intercept).

- Target vector:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- Model prediction:

  $\hat{y} = X\theta$

---

b) Least Squares Objective:

- We want to minimize:

  $J(\theta) = \|y - X\theta\|^2$

- Setting derivative = 0 leads to **Normal Equations**:

  $X^T X\theta = X^T y$

---

c) Solution via Pseudo-inverse:

- If $X^T X$ is invertible:

  $\theta = (X^T X)^{-1} X^T y$

- If $X^T X$ is **not invertible** (e.g., collinear features, more features than data points), we use the **Moore-Penrose Pseudo-inverse**:

  $\theta = X^+ y$

  where X^+ = pseudo-inverse of X.

---

d) Practical Aspects:

- Works well for **small to medium datasets** (analytical closed-form solution).
- Computational cost: O(d^3) for matrix inversion.
- For **large datasets**, iterative methods (like gradient descent) are more efficient.

---

2. Key Takeaways

- Regression coefficients can be solved directly using **matrix algebra**.
- Core formula:
  $\theta = (X^T X)^{-1} X^T y$
- If $X^T X$ is not invertible → use **pseudo-inverse**.
- Advantage: Exact solution, no need for tuning.
- Limitation: Expensive for high-dimensional data → iterative methods preferred.
- Foundation for many ML algorithms — even advanced ones often rely on pseudo-inverse under the hood.

Generalization of Regression Models

1. Detailed Conceptual Overview

So far, we have focused on **fitting a regression to training data**. But in ML, the true test is **how well the model generalizes** — i.e., performs on unseen data.

Generalization = the ability of a regression model to make accurate predictions on **new data not used during training**.

---

a) In-sample vs Out-of-sample Recall

- **In-sample error (training error):** Error on the training set.
- **Out-of-sample error (test/generalization error):** Error on new, unseen data.
- A model may have **very low in-sample error** but **very high out-of-sample error** → classic sign of **overfitting**.

---

b) Bias-Variance Tradeoff (conceptual here):

- **High bias (underfitting):** Model too simple → fails to capture trends (both training and test errors high).
- **High variance (overfitting):** Model too complex → memorizes training data but fails on test data (train error low, test error high).

- Good generalization requires a **balance**.

---

c) Factors Affecting Generalization in Regression:

1. **Model Complexity:**
   - Linear regression assumes a linear relationship.
   - If the true relationship is highly nonlinear, linear regression will underfit.
2. **Number of Features vs Samples:**
   - Too many features with too few samples → overfitting.
   - Dimensionality reduction or regularization helps.
3. **Quality of Data:**
   - Noisy data hurts generalization.
   - Outliers can distort the regression fit.
4. **Validation Strategy:**
   - Cross-validation helps estimate generalization error.
   - Splitting into train/validation/test sets is critical.

---

d) Improving Generalization:

- **Regularization** (penalize large coefficients: Ridge, Lasso).
- **Feature selection** (remove irrelevant features).
- **Data augmentation** or collecting more training samples.
- **Cross-validation** to tune complexity before final testing.

---

2. Key Takeaways

- **Generalization = the ability to perform well on unseen data.**
- **Overfitting:** Low training error, high test error.
- **Underfitting:** Both training and test errors are high.
- Achieving good generalization requires balancing **bias and variance**.

- Strategies: regularization, feature selection, cross-validation, and more data.
- Generalization is the **ultimate goal** of regression and all ML models.

Binary Classification with Linear Regression (0/1 Error vs Squared Error)

---

1. Detailed Conceptual Overview

Linear regression was designed for **predicting continuous values**. But can we use it for **classification** (e.g., spam vs not spam, churn vs no churn)?

Let's explore:

---

a) Idea of Using Linear Regression for Classification

- Suppose labels are binary: $y \in \{0, 1\}$.
- We can try to fit a regression model:
  $h_\theta(x) = \theta^T x$
- Then classify based on a threshold:
  - If $h_\theta(x) \geq 0.5$, predict class **1**.
  - Otherwise, predict class **0**.

This works in some simple cases, but has major issues.

---

b) Error Metrics for Classification

1. **0/1 Error (Classification Error):**
   - Definition: 0 if predicted class = true class, 1 otherwise.
   - Total error = count of misclassified points.
   - Problem: This function is **non-differentiable** → can't be optimized with calculus.

$$\text{Error}_{0/1} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{h_\theta(x_i) \neq y_i\}$$

---

1. **Squared Error (Regression-style Error):**
    - Definition: Treat y as 0/1, and minimize squared error like regression.
    - Cost function:

    $$J(\theta) = \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2$$

    - This is differentiable and solvable.

    Problem:

    - Predictions are not restricted to [0, 1].
    - The fitted line may produce values less than 0 or greater than 1, which don't make sense as probabilities.
    - Leads to poor classification boundaries in practice.

---

c) Why We Don't Use Linear Regression for Classification

- Predictions can fall outside [0, 1].
- The decision boundary may not separate classes well.
- 0/1 error is the correct measure for classification, but it's not optimizable directly.
- Squared error is optimizable, but not ideal for classification.

👉 This is exactly why **Logistic Regression** was developed — it uses the **sigmoid function** to map predictions into [0, 1], and optimizes using **cross-entropy loss**, which works better for classification.

---

2. Key Takeaways

- Linear regression can be **forced into classification**, but it's not a good fit.
- **0/1 error** is the natural metric for classification, but not differentiable.

- **Squared error** is optimizable, but doesn't handle classification properly.
- Linear regression may give outputs outside [0, 1], making interpretation problematic.
- This motivates **Logistic Regression**, which fixes these issues by using the sigmoid and cross-entropy loss.
