Regression

Prof. Hariprasad Kodamana

October 12, 2025

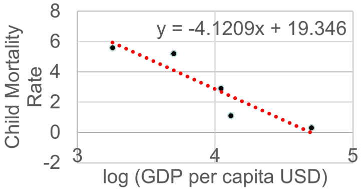
What is Regression?

Definition

Regression analysis is a statistical technique used to model and analyze the relationship between a dependent variable (target) and one or more independent variables (features).

Example:

- Dependent Variable: Energy consumption
- Independent Variables:
  - Temperature
  - Time of day
  - Building occupancy
  - Equipment usage



Types of Regression Problems

Regression Tasks

Predicting Continuous Values

- Energy consumption (kWh)
- Solar panel efficiency (%)
- Equipment temperature (°C)
- Power output (MW)

Common Algorithms:

- Linear Regression
- Polynomial Regression
- Ridge/Lasso Regression

Classification Tasks

Predicting Categories

- Equipment status (On/Off)
- Fault detection (Normal/Faulty)
- Energy efficiency rating (A/B/C)
- Maintenance needed (Yes/No)

Common Algorithms:

- Logistic Regression
- Decision Trees
- Neural Networks

Linear Regression: The Foundation

Mathematical Formulation

Linear regression models the relationship as (hypothesis):

$$h_{\theta(x)} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

But the actual the relationship as (considering uncertainties):

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

$$y = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}^T \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} + \epsilon$$

In matrix form: $y = \theta^T X + \epsilon$

Linear Regression: The Foundation

Components:

- y: Target variable (output)
- x_i: Feature variables (inputs)
- θ_i: Model parameters (weights)
- θ_0: Bias term (intercept)
- ϵ: Error/noise term

Example:

$$\underbrace{\text{Energy Demand (Hyp.)}}_{h_{\theta(x)}} = \theta_0 + \theta_1 \cdot \underbrace{\text{Temperature}}_{x_1} + \theta_2 \cdot \underbrace{\text{Hour}}_{x_2}$$

$$\underbrace{\text{Energy Demand (Actual)}}_{y} = \theta_0 + \theta_1 \cdot \underbrace{\text{Temperature}}_{x_1} + \theta_2 \cdot \underbrace{\text{Hour}}_{x_2} + \epsilon$$

Where:

- θ_1: Temperature sensitivity
- θ_2: Time-of-day effect

Real-World Example: Child Mortality vs GDP

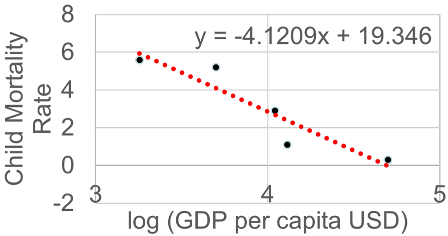
Country	**A**	**B**	**C**	**D**	**E**
GDP per capita (USD)	1800	5000	11000	13000	50000
Child Mortality Rate	5.6	5.2	2.9	1.1	0.3

Research Question: How does economic development affect child health outcomes?

Model:

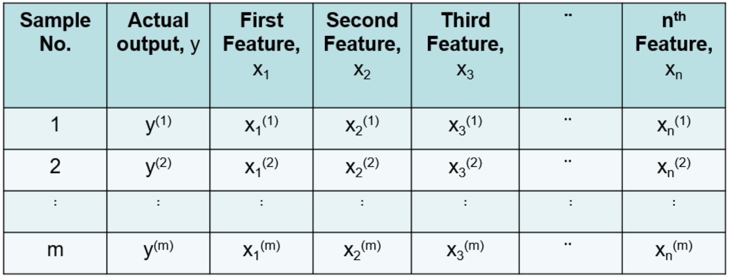
$$\text{CMR} = \theta_0 + \theta_1 \log(\text{GDP})$$

Why logarithm? Economic effects often show diminishing returns



Linear Regression

Typically, m data points are present (each data point is a row with $y, x_1, x_2, ..., x_n$ in a Table) such that $m > n$.



Sample No.	Actual output, y	First Feature, x_1	Second Feature, x_2	Third Feature, x_3	..	n^{th} Feature, x_n
1	$y^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	..	$x_n^{(1)}$
2	$y^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	..	$x_n^{(2)}$
:	:	:	:	:	:	:
m	$y^{(m)}$	$x_1^{(m)}$	$x_2^{(m)}$	$x_3^{(m)}$	..	$x_n^{(m)}$

Objective is to obtain the optimal values of weights, θ, which gives the best fit of the model output y' with given output y. Clearly, the error between y and y' need to be minimized. Typically, the square of error is minimized and the optimum RMSE or MSE is reported.

Linear Regression

Vector-Matrix Structure of Linear Regression

$$\underbrace{\begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(m) \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & X_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}}_{\theta} + \underbrace{\begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_m \end{bmatrix}}_{\boldsymbol{\epsilon}}$$

How Do We Find the Best Model?

The Optimization Problem

Find parameters θ that minimize prediction errors:

$$J(\theta) = \min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - h^{(i)}_{\theta(x)})^2 = \min_{\theta} \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \theta^{T(i)} X)^2$$

Two Main Approaches:

1. Analytical Solution

Normal Equation:

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

Pros: Exact solution, no iterations
Cons: Slow for large datasets, requires matrix inversion

2. Iterative Solution

Gradient Descent:

$$\theta_{k+1} := \theta_k - \alpha \nabla J(\theta_k)$$

Pros: Scales to large data, always works **Cons:** Requires tuning, many iterations

Analytical Solution: Normal Equation

Mathematical Derivation **Step 1:** Define cost function

$$J(\theta) = \frac{1}{2m}(\mathbf{y} - \mathbf{X}\theta)^T(\mathbf{y} - X\theta)$$

Step 2: Expand

$$J(\theta) = \frac{1}{2m}(\mathbf{y}^T\mathbf{y} - 2\theta^T\mathbf{X}^T\mathbf{y} + \theta^T\mathbf{X}^T\mathbf{X}\theta)$$

$$\frac{\partial}{\partial \mathbf{X}}(\mathbf{a}^T\mathbf{X}) = \mathbf{a}, \frac{\partial}{\partial \mathbf{X}}(\mathbf{a}) = \mathbf{0}, \frac{\partial}{\partial \mathbf{X}}(\mathbf{a}^T\mathbf{X}\mathbf{a}) = 2\mathbf{X}\mathbf{a} \text{ If } \mathbf{X} \text{ is symmetric}$$

Step 3: Take derivative and set to zero

$$\frac{\partial J}{\partial \theta} = \frac{1}{m}(-\mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\theta) = 0$$

Step 4: Solve for θ

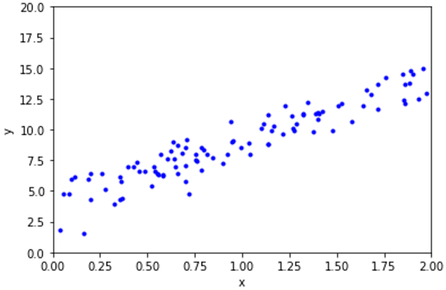
$$\theta^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

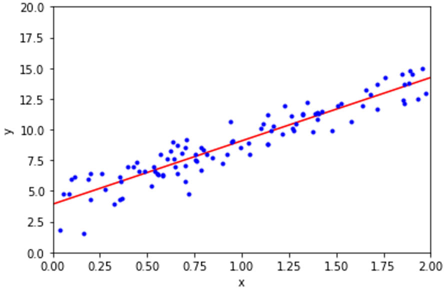
When to Use

Best for small to medium datasets

Linear Regression

1. To illustrate the concept, consider a simple example of fitting the data as $y = \theta_0 + \theta_1 x$
2. Artificially, the data of 100 size is generated as $x = 2 \times \text{rand}()$; $y = 4 + 5x + \text{rand}()$





Best fit is obtained as: $y = 3.9229 + 5.1540x$

Why Do We Need Optimization?

When Closed Form Solutions Fail

- Large datasets ($X^T X$ becomes computationally expensive)
- Non-linear models (no analytical solution exists)
- Regularized models (Ridge, Lasso)
- Real-time applications (need iterative updates)

Solution: Iterative Optimization

Instead of solving analytically, we use iterative algorithms to find the minimum of the cost function.

Gradient Descent is the Foundation

"All machine learning optimizers are extensions of gradient descent"

Gradient Descent Algorithm

Core Concept

Start with initial parameters and iteratively move in the direction of steepest descent:

$$\theta_{k+1} = \theta_k - \alpha \nabla J(\theta_k)$$

where α is the learning rate and $\nabla J(\theta)$ is the gradient.

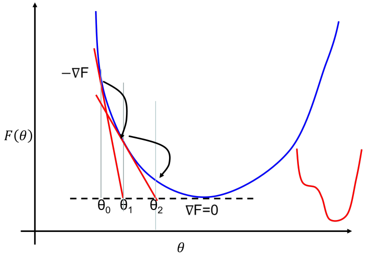
Algorithm Steps:

1. Initialize θ randomly
2. Compute gradient $\nabla J(\theta)$
3. Update: $\theta := \theta - \alpha \nabla J(\theta)$
4. Repeat until convergence

For Linear Regression:

$$\nabla J(\theta) = \mathbf{X}^T(\mathbf{X}\theta - \mathbf{y}) = \mathbf{X}^T(h_{\theta(\mathbf{x})} - \mathbf{y})$$

$$\nabla J^{(i)}(\theta) = (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$



Optimizers Used in Machine Learning

1. Gradient Descent
2. Nesterov Accelerated Gradient (NAG)
3. Adaptive Gradient Algorithm (AdaGrad)
4. Root Mean Square Propagation (RMS-Prop)
5. Adaptive Moment Estimation (Adam)

All Machine learning optimizers used are the extension of Gradient Descent.

Gradient Descent

1. Vanilla/ Batch Gradient Descent: Calculates the averaged gradient with all training samples.
2. Stochastic Gradient Descent: Calculates the gradient with one training sample. Response is memory efficient but unstable (response fluctuates) and may shoot even after getting global optimum. Concept of momentum reduces the fluctuations.
3. Mini Batch Gradient Descent: The data is divided into different mini batches. Gradient is calculated as average value over all samples present in one mini batch.

Trade-offs

Batch: Stable but slow • **SGD:** Fast but noisy • **Mini-batch:** Best of both worlds

GD for Linear Regression

- Batch Gradient Descent
  Uses the entire training set of m samples for each update.

$$\theta_{k+1} = \theta_k - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

- Stochastic Gradient Descent (SGD)
  Uses a single sample i for each update.

$$\theta_{k+1} = \theta_k - \alpha \cdot (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

- Mini-batch Gradient Descent
  Uses a mini-batch B of b samples for each update.

$$\theta_{k+1} = \theta_k - \alpha \cdot \frac{1}{b} \sum_{i \in B} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

Epochs and Gradient Descent Variants

Definition of Epoch

One epoch = One complete pass through the entire training dataset

Batch Gradient Descent

$$\theta_{k+1} = \theta_t - \alpha \nabla J(\theta_k; \mathcal{D}_{\text{full}})$$

- Uses entire dataset for each update
- 1 epoch = 1 parameter update
- Stable convergence
- Memory intensive
- Slow for large datasets

Stochastic Gradient Descent (SGD)

$$\theta_{k+1} = \theta_k - \alpha \nabla J(\theta_k; x^{(i)})$$

- Uses single sample for each update
- 1 epoch = N parameter updates (N = dataset size)
- Noisy convergence
- Memory efficient
- Fast updates

Mini-Batch Gradient Descent

$$\theta_{k+1} = \theta_k - \alpha \nabla J(\theta_k; \mathcal{B})$$

- Uses small batch of samples (e.g., 32, 64, 128)
- 1 epoch = N/B parameter updates (B = batch size)
- Balanced approach
- Good for parallelization
- Most commonly used

Comparison: Updates per Epoch

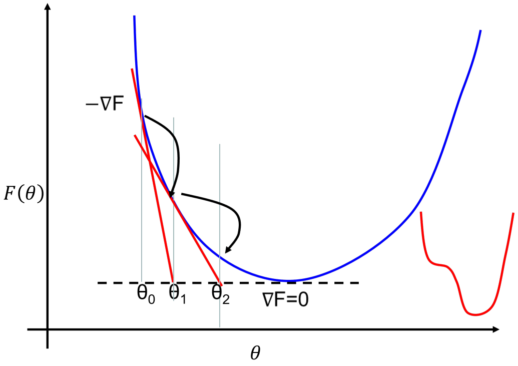
	Batch Size	Updates/Epoch	Convergence
Batch GD	N	1	Smooth
Mini-Batch	B	$\lceil N/B \rceil$	Moderate
SGD	1	N	Noisy

Where: N = total samples, B = batch size

Key Relationships

- Epochs measure progress through full dataset
- Batch size affects update frequency per epoch
- Mini-batch offers best trade-off for most cases

Gradient Descent with Momentum



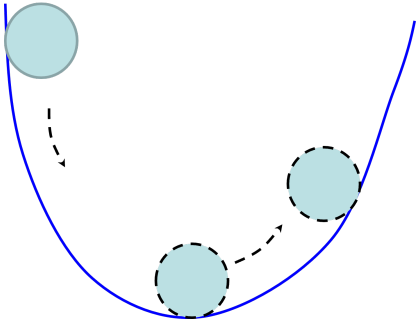
Concept of Momentum

$$v_k = \gamma v_{k-1} + \alpha \nabla J(\theta_k)$$
$$\theta_{k+1} = \theta_k - v_k$$

1. A typical value of γ used is 0.9.
2. Concept is like rolling a ball from a hill to valley while finding the minima.

Nesterov Accelerated Gradient (NAG)

1. Requires a more efficient way than just rolling a ball from hill to valley while finding the minimum. When we reach the minimum there is a need to slow down otherwise the ball will not stop at flat surface at the minimum and will continue to move up.
2. NAG provides an ability of slowing down when algorithm reaches close to minimum.
3. It works better than the conventional momentum algorithm.



Gradient is calculated with respect to estimated future position

$$\theta_{k+1,\text{estimated}} = \theta_k - \gamma v_{k-1}$$

$$v_k = \gamma v_{k-1} + \alpha \nabla J(\theta_{k+1,\text{estimated}})$$

$$\theta_{k+1} = \theta_k - v_k$$

Adaptive Gradient Algorithm (AdaGrad)

1. It adapts different learning rates for different parameters.
2. The learning rate decreases with increase in number of steps.
3. It is well suited for cases in which data is sparse.

$$
\begin{aligned}
g_{0,i} &= 0 \\
g_{k,i} &= g_{k-1,i} + [\nabla J(\theta_{k,i})]^2 \\
\theta_{k+1,i} &= \theta_{k,i} - \frac{\alpha \nabla J(\theta_{k,i})}{\sqrt{g_{k,i} + \epsilon}}
\end{aligned}
$$

4. It does not require the tuning of learning rate. α is kept at 0.01.
5. Accumulation of positive square gradients makes $g_{k,i}$ very large, which makes the learning rate very small over the iterations.

AdaGrad's aggressively decreasing learning rate is reduced in these methods. The gradient accumulation is modified with a decay rate γ (typically 0.9):

$$g_{k,i} = \gamma g_{k-1,i} + (1-\gamma)[\nabla J(\theta_{k,i})]^2$$

AdaDelta:

$$\theta_{k+1,i} = \theta_{k,i} - \frac{\sqrt{g_{k-1,i} + \epsilon}}{\sqrt{g_{k,i} + \epsilon}} \nabla J(\theta_{k,i})$$

RMS-Prop:

$$\theta_{k+1,i} = \theta_{k,i} - \frac{\alpha \nabla J(\theta_{k,i})}{\sqrt{g_{k,i} + \epsilon}}$$

Adaptive Moment Estimation (Adam)

1. It adapts different learning rates for different parameters. In addition to storing of square of gradient for calculating the learning rate (similar to RMS-Prop), it also adapts the gradient with the concept of momentum.

$$
\begin{aligned}
m_{0,i} &= 0, \quad g_{0,i} = 0 \\
m_{k,i} &= \beta_1 m_{k-1,i} + (1-\beta_1)\nabla J(\theta_{k,i}) \\
g_{k,i} &= \beta_2 g_{k-1,i} + (1-\beta_2)[\nabla J(\theta_{k,i})]^2 \\
M_{k,i} &= \frac{m_{k,i}}{(1-\beta_1)}; \quad G_{k,i} = \frac{g_{k,i}}{(1-\beta_2)} \\
\theta_{k+1,i} &= \theta_{k,i} - \frac{\alpha}{\sqrt{G_{k,i}+\epsilon}} M_{k,i}
\end{aligned}
$$

2. Typical values used are $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.
3. Selection of the optimizer depends on the data. Typically for sparse data, adaptive learning methods are best. Further, if learning rate is diminishing fast, then the methods such as RMS-Prop, AdaDelta and Adam are good. Overall, Adam is best among all.

Optimization Algorithms: Momentum, NAG, and AdaGrad

Momentum

$$v_k = \gamma v_{k-1} + \alpha \nabla J(\theta_k)$$
$$\theta_{k+1} = \theta_k - v_k$$

- $\gamma \approx 0.9$ (typical)
- Accumulates gradient history
- Reduces oscillations in ravines
- Like rolling ball downhill

NAG (Nesterov)

$$\theta_{k+1,\text{est}} = \theta_k - \gamma v_{k-1}$$
$$v_k = \gamma v_{k-1} + \alpha \nabla J(\theta_{k+1,\text{est}})$$
$$\theta_{k+1} = \theta_k - v_k$$

- "Lookahead" gradient
- Slows down near minimum
- More stable convergence
- Better than Momentum

AdaGrad

$$g_{k,i} = g_{k-1,i} + [\nabla J(\theta_{k,i})]^2$$
$$\theta_{k+1,i} = \theta_{k,i} - \frac{\alpha \nabla J(\theta_{k,i})}{\sqrt{g_{k,i} + \epsilon}}$$

- $\epsilon \approx 10^{-8}$, $\alpha \approx 0.01$
- Adaptive learning rates
- Good for sparse data
- Learning rate decreases over time

Key Characteristics

- **Momentum**: Velocity-based, handles saddle points well
- **NAG**: Gradient computed at estimated future position, better convergence
- **AdaGrad**: Per-parameter adaptive learning rates, no learning rate tuning needed
- All use: α = learning rate, $\nabla J(\theta_k)$ = gradient at step k

Adaptive Optimization Algorithms: AdaDelta, RMSprop, and Adam

RMSprop

$$g_k = \gamma g_{k-1} + (1-\gamma)[\nabla J(\theta_k)]^2$$

$$\theta_{k+1} = \theta_k - \frac{\alpha \nabla J(\theta_k)}{\sqrt{g_k + \epsilon}}$$

- $\gamma = 0.9$, $\epsilon = 10^{-8}$
- Exponential decay
- No manual α tuning

AdaDelta

$$g_k = \gamma g_{k-1} + (1-\gamma)[\nabla J(\theta_k)]^2$$

$$\Delta\theta_k = \frac{\sqrt{\Delta\theta_{k-1} + \epsilon}}{\sqrt{g_k + \epsilon}} \nabla J(\theta_k)$$

$$\theta_{k+1} = \theta_k - \Delta\theta_k$$

- No learning rate specified
- Uses update history
- Completely adaptive

Adam

$$m_k = \beta_1 m_{k-1} + (1-\beta_1)\nabla J(\theta_k)$$

$$g_k = \beta_2 g_{k-1} + (1-\beta_2)[\nabla J(\theta_k)]^2$$

$$\hat{m}_k = \frac{m_k}{1-\beta_1^k}$$

$$\hat{g}_k = \frac{g_k}{1-\beta_2^k}$$

$$\theta_{k+1} = \theta_k - \frac{\alpha}{\sqrt{\hat{g}_k + \epsilon}} \hat{m}_k$$

- $\beta_1 = 0.9$, $\beta_2 = 0.999$
- Momentum + adaptive
- Bias correction

Key Characteristics

- **RMSprop**: Fixed α with adaptive gradients, prevents aggressive decay
- **AdaDelta**: Learning-rate free, uses moving window of gradient updates
- **Adam**: Combines momentum + adaptive learning + bias correction, best overall
- All: Exponential decay, suitable for sparse data and non-convex problems

ADAM

Adam: A Hybrid Approach

Adam combines the best features of:

- RMSprop: Adaptive learning rates using squared gradients

$$g_k = \beta_2 g_{k-1} + (1-\beta_2)[\nabla J(\theta_k)]^2$$

- Momentum: First-moment estimation for acceleration

$$m_k = \beta_1 m_{k-1} + (1-\beta_1)\nabla J(\theta_k)$$

Adam's Unique Enhancements:

- Bias Correction: Compensates for initialization bias

$$\hat{m}_k = \frac{m_k}{1-\beta_1^k}, \quad \hat{g}_k = \frac{g_k}{1-\beta_2^k}$$

- Combined Optimization: Integrated update rule

$$\theta_{k+1} = \theta_k - \frac{\alpha}{\sqrt{\hat{g}_k + \epsilon}}\hat{m}_k$$

Algorithm Feature Comparison

	Momentum	Adaptive LR	Bias Correction
RMSprop	×	✓	×
AdaDelta	×	✓	×
Adam	✓	✓	✓

Key Constants and Typical Values:

- Learning rate: $\alpha = 0.001$ (default)
- First moment decay: $\beta_1 = 0.9$
- Second moment decay: $\beta_2 = 0.999$
- Numerical stability: $\epsilon = 10^{-8}$
- RMSprop decay: $\gamma = 0.9$

Key Advantages:

- Faster convergence with momentum
- Adaptive learning rates per parameter
- Bias correction for stable early training
- Robust default hyperparameters

Adam = Momentum (β_1) + RMSprop (β_2) + Bias Correction

Closed-Form vs. Gradient Descent: A Comparison

Closed-Form Solution

- **Pro:** No need for learning rate tuning. No iterations.
- **Con:** Slow for many features (n). Inverting (X^TX) is computationally expensive ($O(n^3)$).
- **Con:** **Fails** if X^TX is singular (non-invertible). This happens with redundant features or when $n > m$.

Gradient Descent

- **Pro:** Scales well to millions of features.
- **Pro:** Works even if X^TX is singular.
- **Con:** Requires choosing a good learning rate α.
- **Con:** Requires many iterations to converge.

For most large-scale energy problems, Gradient Descent and its variants are the methods of choice.

Linear Regression Assumptions

Critical Assumptions for Valid Results

The validity of linear regression depends on several key assumptions:

1. **Linearity:** The relationship between variables is linear

$$E(y|X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

2. **Independence:** Observations are independent of each other

$$\text{Cov}(\epsilon_i, \epsilon_j) = 0 \text{ for } i \neq j$$

3. **Homoscedasticity:** Constant variance of residuals

$$\text{Var}(\epsilon_i|X) = \sigma^2$$

Linear Regression Assumptions

4. **Normality:** Residuals are normally distributed

$$\epsilon_i \sim N(0, \sigma^2)$$

5. **No Multicollinearity:** Independent variables are not highly correlated

$$(\textit{Varinance Inflaction Factor})\ \text{VIF}_i = \frac{1}{1 - R_i^2} < 5 \text{ (rule of thumb)}$$

Checking Assumptions in Energy Data

- **Linearity:** Residual plots should show random scatter
- **Independence:** Check for autocorrelation in time-series data
- **Homoscedasticity:** Residuals vs. fitted values plot
- **Normality:** Q-Q plots of residuals
- **Multicollinearity:** VIF analysis for correlated features (e.g., temperature and cooling demand)

Model Evaluation I: Mean Absolute Error (MAE)

What is MAE?

MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It is the average of the absolute differences between prediction and actual observation.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - h_{\theta(x)i}|$$

Interpretation and Properties

- **Interpretation:** MAE is in the same units as the original target variable, making it easy to interpret. An MAE of 5.5 MW in an energy forecast means the predictions are, on average, off by 5.5 MW.
- **Key Property:** Because it uses the absolute value, it does not penalize large errors disproportionately. This makes it a **robust** metric that is not sensitive to outliers.

Model Evaluation II: Root Mean Squared Error (RMSE)

What is RMSE?

RMSE is the square root of the average of the squared differences between the prediction and the actual observation. It represents the standard deviation of the residuals (prediction errors).

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - h_{\theta(x)i})^2}$$

Interpretation and Properties

- **Interpretation:** Like MAE, RMSE is also in the original units of the target variable.
- **Key Property:** Due to the squaring, RMSE gives a relatively high weight to large errors. This means it is more useful when large errors are particularly undesirable. An occasional large error will have a significant effect on the RMSE.

Model Evaluation III: R-squared (R^2)

What is R-squared?

Also known as the Coefficient of Determination, R^2 is a statistical measure of how well the regression predictions approximate the real data points. It represents the proportion of the variance in the dependent variable that is predictable from the independent variables.

$$R^2 = 1 - \frac{\sum(y_i - h_{\theta(x)i})^2}{\sum(y_i - \bar{y})^2} = 1 - \frac{\text{Unexplained Variance}}{\text{Total Variance}}$$

Interpretation and Limitations

- **Interpretation:** An R^2 of 0.8 means that 80% of the variance in the outcome can be explained by the model's inputs.
- **Limitation:** R^2 **will always increase** if you add more predictors to the model, even if those predictors are unrelated to the outcome. This can make it a misleading metric for model comparison.

Model Evaluation IV: Adjusted R-squared

Why do we need a better R^2?

Because the standard R^2 can be artificially inflated by simply adding more variables, we need a metric that penalizes a model for being overly complex.

Adjusted R-squared

This is a modified version of R^2 that has been adjusted for the number of predictors in the model. It only increases if the new predictor improves the model more than would be expected by chance.

$$\text{Adjusted } R^2 = 1 - \left[\frac{\mathrm{SS}_{\mathrm{res}}/(n-p-1)}{\mathrm{SS}_{\mathrm{tot}}/(n-1)} \right]$$

(where n is the number of data points and p is the number of predictors).

The Rule of Thumb Always use **Adjusted R-squared** when comparing the goodness-of-fit between regression models that have different numbers of predictors. It is a much more reliable metric for model selection.

R-squared: The Most Important Metric

What Does R^2 Really Mean?

$$R^2 = 1 - \frac{\text{Sum of Squared Residuals}}{\text{Total Sum of Squares}} = 1 - \frac{SS_{res}}{SS_{tot}}$$

Interpretation Guide:

- $R^2 = 1.0$: Perfect prediction (model explains all variance)
- $R^2 = 0.8$: Good model (explains 80
- $R^2 = 0.5$: Moderate model (explains 50
- $R^2 = 0.0$: Model is no better than using the mean
- $R^2 < 0.0$: Model is worse than using the mean!

Adjusted R^2: The Better Version

$$R^2_{adj} = 1 - \frac{(1 - R^2)(n - 1)}{n - p - 1}$$

Penalizes models for having too many features. Always use this for model comparison!

Why Adjusted R^2? The Degrees of Freedom Story

The Problem with R^2

- Always increases when adding predictors
- Even for **useless** variables!
- Leads to **overfitting**

The Solution: Adjusted R^2

- **Penalizes** adding useless predictors
- Uses **degrees of freedom** in denominator

Degrees of Freedom Matter

- **Residual df:** $n - k - 1$ (what's left after fitting)
- **Total df:** $n - 1$ (around the mean)
- Small **residual df** = Large **penalty**

Key Insight

Adjusted R^2 only increases if new predictors improve model **more than expected by chance**!

Locally Weighted Linear Regression

1. Objective is to fit the model preferentially at local region. For instance, if a large range of data is given, and one want to fit the linear model preferentially around a selected data point.:

$$\min MSE(\theta) = \frac{1}{m}\sum_{i=1}^{m} w^{(i)}(\theta^T x^{(i)} - y^{(i)})^2$$

   Where, $w^{(i)}$ are the weights corresponding to the data points.

2. If $w^{(i)}$ is large, the penalization of $(\theta^T x^{(i)} - y^{(i)})^2$ is large and when $w^{(i)}$ is small the penalization of $(\theta^T x^{(i)} - y^{(i)})^2$ is small.
3. The standard choice of $w^{(i)}$ at a particular query point is:

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

4. For small $|x^{(i)} - x|$, $w^{(i)} \approx 1$. For large $|x^{(i)} - x|$, $w^{(i)} \approx 0$.
5. The bandwidth parameter τ which decides how the weight of a training example falls off with distance of its $x^{(i)}$ from the query point x.

Polynomial Regression

1. Often, the data is complex than that represented by a straight line. Fitting such data using a polynomial is referred as Polynomial regression.
2. Interesting to note that the linear regression can be used for representing such complex data by representing the 'feature with powers' by new features and then taking a linear combinations of these.

$$h'_{\theta(z)} = \theta_0 + \theta_1 z_1 + \theta_2 z_2^2 + \cdots + \theta_n z_n^n = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

$$x_1 = z_1, x_2 = z_2^2, \ldots, x_n = z_n^n$$

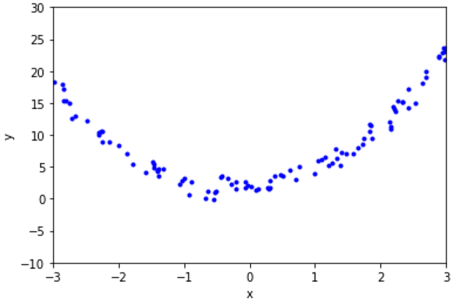
3. A polynomial of any degree for given number of features can be given as input to fit the data. For instance, if degree = 3 and features = 2 (a, and b) then the output will be

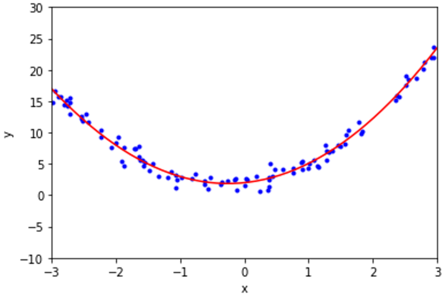
$$h_{\theta(x)} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1 x_2^2 + \theta_6 x_1^3 + \theta_7 x_2^3$$

4. Challenge is to obtain correct degree of the polynomial which do not cause underfitting or overfitting.

Polynomial Regression

1. For illustration, consider fitting $h_{\theta(x)} = \theta_0 + \theta_1 x + \theta_2 x^2$ to the data.
2. Artificially, the data of 100 size is generated as x = 6×rand() - 3; y = 2 + x + 2x^2 + rand()

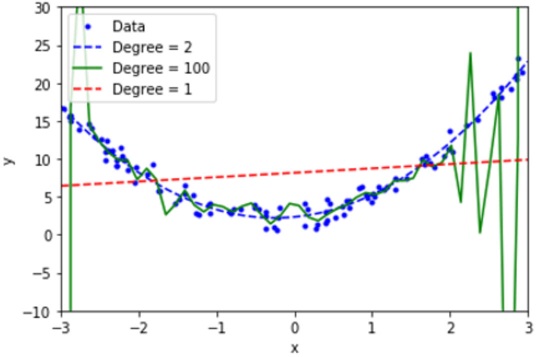




Best fit is obtained as: $y = 1.775 + 0.9763x + 2.02x^2$

Polynomial Regression - Over fitting

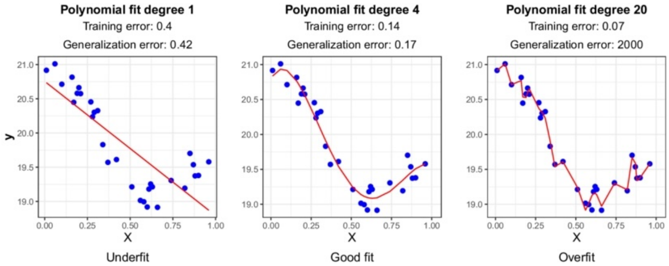
1. With increase in degree, the polynomial curve tries to fit every data point resulting into wiggling of the model.



1. Most appropriate fitting occurs with degree $= 2$ as we know that the data is generated with degree $= 2$. However, in real life case this information is not known and one has to evaluate the most appropriate value of degree which do not cause overfitting or underfitting.

Over fitting, Perfect fitting and Under fitting

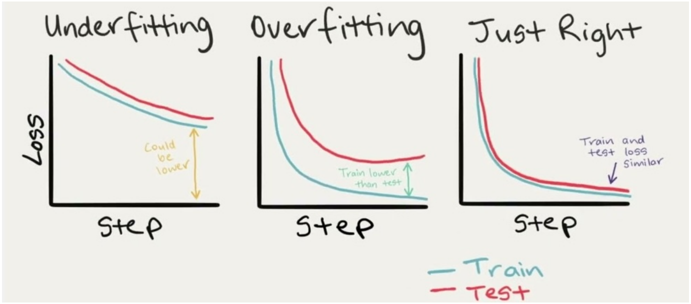
1. To evaluate whether the model is overfitting or under-fitting is extremely important question.
2. One can evaluate whether the model is overfitting or underfitting from learning curves.



1. When settling error for both training and validation is high then it shows underfitting.
2. When the gap between training and validation curves is large then it shows overfitting.

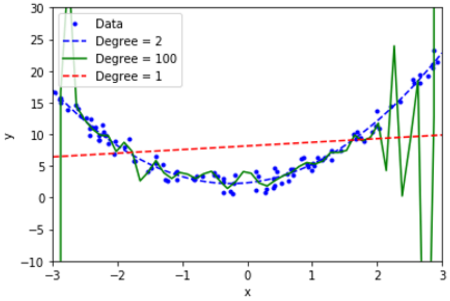
Over fitting, Perfect fitting and Under fitting

1. To evaluate whether the model is overfitting or under-fitting is extremely important question.
2. One can evaluate whether the model is overfitting or underfitting from learning curves.



1. When settling error for both training and validation is high then it shows underfitting.
2. When the gap between training and validation curves is large then it shows overfitting.

The Danger of Overfitting



The Problem:

- Higher degree = more complex model
- Model tries to fit every data point
- Results in "wiggly" predictions
- Poor generalization to new data

The Solution:

- Choose degree carefully
- Use validation data
- Apply regularization
- Cross-validation

Key Insight

The goal is not to fit training data perfectly, but to generalize well to new, unseen data!

The Fundamental Challenge: Bias-Variance Tradeoff

Model Error Decomposition

Total Error $=$ Bias² $+$ Variance $+$ Irreducible Error

High Bias (Underfitting)

Problem: Model too simple

- Makes strong assumptions
- Misses underlying patterns
- Poor performance on both training and test data

Example: Using linear model for wind power (which follows v^3 relationship)

High Variance (Overfitting)

Problem: Model too complex

- Learns noise in training data
- Perfect on training, poor on test
- Small changes in data cause large changes in model

Example: 15th degree polynomial for simple relationship

Goal: Find the sweet spot that minimizes total error!

Model Validation: Avoiding Overfitting

The Problem

How do we estimate model performance on unseen data without using our final test set?

Data Splitting Strategy:

- **Training Set (60-70%):** Train model parameters
- **Validation Set (15-20%):** Tune hyperparameters & select models
- **Test Set (15-20%):** Final, unbiased performance estimate

Golden Rule

Never use test data for model selection or tuning. Save it for the final evaluation only!

Energy Example

- Training: 2018-2020 energy consumption data
- Validation: 2021 data (tune model complexity)
- Test: 2022 data (final performance)

K-Fold Cross-Validation: Robust Model Evaluation

The Problem with Simple Validation

Performance can depend heavily on how we split the data. One "unlucky" split can mislead us.

K-Fold Cross-Validation Algorithm:

1. Divide training data into K equal folds (typically K=5 or 10)
2. For each fold i = 1 to K:
    - Train model on K-1 folds
    - Test on fold i
    - Record performance
3. Average performance across all K folds

Advantages:

- More robust performance estimate
- Uses all data for both training and validation
- Reduces impact of data split randomness
- Industry standard for model evaluation

Energy Applications:

- Solar farm performance modeling
- Energy demand forecasting
- Grid stability prediction
- Equipment failure detection

Fighting Overfitting: Regularization

The Core Idea

Add a penalty term to the cost function to discourage overly complex models.

Why Regularization Works:

- Large coefficients often indicate overfitting
- Penalty forces model to balance fit quality vs. simplicity
- Helps model generalize better to new data

General Form:

$$J_{regularized}(\theta) = J_{original}(\theta) + \alpha \cdot \text{Penalty}(\theta)$$

Where:

- α: Regularization strength (hyperparameter)
- Higher α = more penalty = simpler model
- Lower α = less penalty = more complex model

Three Main Types

Ridge (L2), Lasso (L1), and Elastic Net (combination)

Ridge Regression (L2 Regularization)

1. It represents a regularized version of linear regression.
2. It forces not only fitting of the data but also keep the weights as small as possible.
3. It is to be noted that the regularization term is added (half of the square of the L_2 norm of the weight vector) only for the training of the model. Once the trained model is obtained, it is fit to the validation data without the regularization terms to evaluate its performance.

$$\min J(\theta) = \text{MSE}(\theta) + \underbrace{\alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2}_{\text{Regularization term}}$$

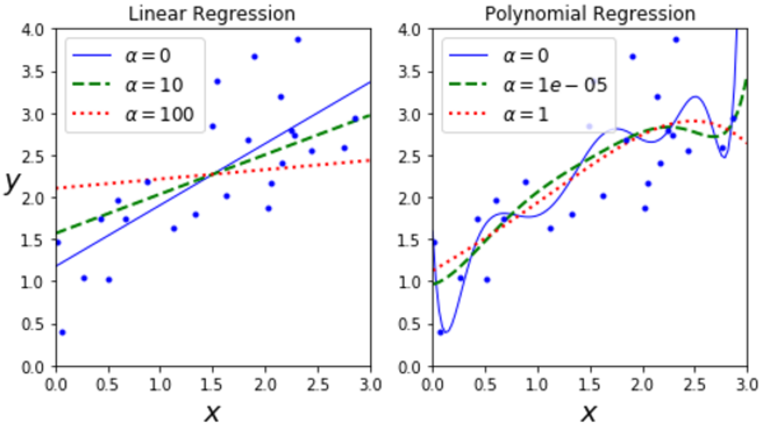
In matrix form Regularization term$=\frac{1}{2}\alpha\theta^T \mathbf{I}\theta$, where $\mathbf{I}$ is (n+1) $\times$ (n+1) identity matrix except with 0 term in the top left for bias term, and $\theta = [\theta_0, \ldots, \theta_n]^T$

4. Note that the bias term θ_0 is not regularized and α is suitable scaling constant.
5. The closed form solution is given by:

$$\theta^* = (X^T X + \alpha I)^{-1} X^T y$$

Ridge Regression (L2 Regularization)

The solution adds a positive constant to the diagonal of $X^T X$ before inversion. This makes the problem nonsingular, even if $X^T X$ is not of full rank, and was the main motivation for ridge regression.



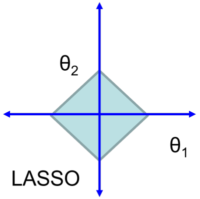
Increase in α values flattens the curves and increases the bias

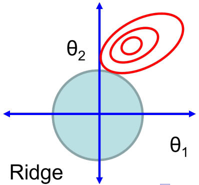
LASSO Regression (L1 Regularization)

1. It represents another regularized version of linear regression similar to ridge regression.

$$\min J(\theta) = \text{MSE}(\theta) + \underbrace{\alpha \sum_{i=1}^{n} |\theta_i|}_{\text{Regularization term}}$$

2. Note that the bias term θ_0 is not regularized.
3. L_1 norm of the weight vector is added for regularizing the MSE. This L_1 penalty shrinkage is much more than that obtained using half of the square of L_2.
4. This latter constraint makes the solution non-linear in $y^{(i)}$ and there is no closed form solution possible for LASSO similar to Ridge regression.





LASSO Regression (L1 Regularization)

1. For the simple case having two parameters θ_1 and θ_2, the residual sum of squares has elliptical contours, centered at the full least squares estimate.
2. Constraint for ridge regression is disk:

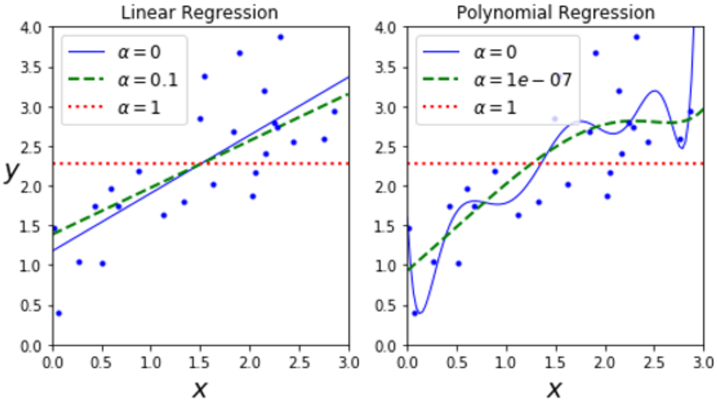
$$\theta_1^2 + \theta_2^2 \leq t$$

3. Constraint for LASSO regression is diamond:

$$|\theta_1| + |\theta_2| \leq t$$

4. In constraint optimization, solution would lie at the active constraints (on the boundary of the constraint surface).
5. Unlike the disk, the diamond has corners; if the solution occurs at a corner, then it will make one parameter θ_j equal to zero.
6. Thus, shrinkage in Lasso is more than in Ridge.

LASSO Regression (L1 Regularization)



Increase in α values flattens the curves and increases the bias

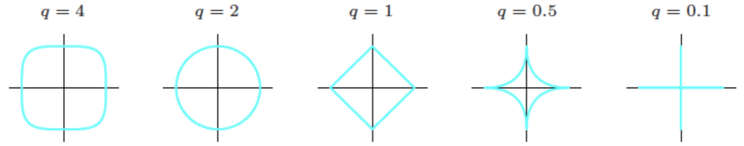
Generalization of LASSO

1. The objective function is modified in the generalized form of LASSO:

$$\min J(\theta) = \text{MSE}(\theta) + \underbrace{\alpha \sum_{i=1}^{n} ||\theta_i||_q}_{\text{Regularization term}}$$

where $\|\theta\|_q := \left(\sum_{i=1}^{n} |\theta_i|^q \right)^{1/q}$.

2. The value q $=$ 0 corresponds to variable subset selection, as the penalty simply counts the number of nonzero parameters.
3. q $=$ 1 corresponds to the Lasso.
4. q $=$ 2 correspond to ridge regression.



Elastic Net Regression

1. The elastic net is a regularized regression method that linearly combines the L_1 and L_2 penalties of the Lasso and Ridge methods.
2. The objective function is modified as follows:

$$\min J(\theta) = MSE(\theta) + r\alpha \sum_{i=1}^{n} |\theta_i| + (1-r)\alpha \sum_{i=1}^{n} \theta_i^2$$

3. $r = 1$ corresponds to the Lasso.
4. $r = 0$ correspond to ridge regression.

Beyond Regression: Classification Problems

When the Output is Categorical

Not all problems predict continuous values. Sometimes we need to classify:

- Equipment status: Working / Faulty
- Energy source: Renewable / Non-renewable
- Grid condition: Stable / Unstable

Why Linear Regression Fails for Classification:

- Outputs continuous values, but we need discrete categories
- No natural way to handle probability constraints
- Poor decision boundaries

Enter Logistic Regression:

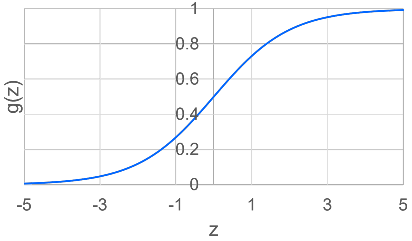
- Outputs probabilities (0 to 1)
- Built for binary classification
- Uses same optimization techniques as linear regression
- Natural extension to our regression toolkit

Logistic Regression

1. Logistic regression aims at solving binary classification problem in which output y can take on only two values, 0 and 1.
2. Since linear regression gives continuous valued output, it performs poorly for such classification problem in which desired output is discrete.

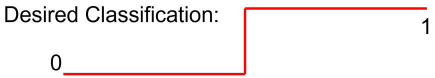
Logistic Regression Model: $h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$

Where, $g(z) = \frac{1}{1+e^{-z}}$ is called logistic or sigmoid function



When $z \to \infty$, $g(z) \to 1$
When $z \to -\infty$, $g(z) \to 0$

Sigmoidal function gives classification close to desired binary classification



Logistic Regression

1. g(z) has an interesting property:

$$
\begin{aligned}
g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} = -\frac{1}{(1+e^{-z})^2}(-e^{-z}) \\
&= \frac{1}{(1+e^{-z})}\frac{e^{-z}}{(1+e^{-z})} = \frac{1}{(1+e^{-z})}\left(1-\frac{1}{(1+e^{-z})}\right) \\
&= g(z)(1-g(z))
\end{aligned}
$$

2. Let us assume that:

$$p(y=1|x;\theta) = h_\theta(x)$$

$$p(y=0|x;\theta) = 1 - h_\theta(x)$$

3. Combining these: $p(y|x;\theta) = (h_\theta(x))^y(1-h_\theta(x))^{(1-y)}$
4. MLE: If there are m independent examples:

$$L(\theta) = p(Y|X;\theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta) = \prod_{i=1}^{m}(h_\theta(x^{(i)}))^{y^{(i)}}(1-h_\theta(x^{(i)}))^{(1-y^{(i)})}$$

The Principle of Maximum Likelihood Estimation (MLE)

The Core Question

Given the data we have observed, what are the most plausible values for our model parameters (θ)?

The Likelihood Function $L(\theta)$

The likelihood function answers this by treating the model parameters θ as variables. It measures how "likely" our observed data is for any given set of parameters.

$$L(\theta) = p(Y|X;\theta) = \prod_{i=1}^{m} p(y^{(i)}|\mathbf{x}^{(i)};\theta)$$

Our goal is to find the θ that **maximizes** this likelihood.

The Principle of Maximum Likelihood Estimation (MLE)

The Strategy: Maximizing the Log-Likelihood

- The product of many small probabilities can become numerically unstable (vanishingly small).
- Therefore, we maximize the **log-likelihood** instead, which turns the product into a sum and is easier to differentiate.

$$l(\theta) = \ln L(\theta) = \sum_{i=1}^{m} \ln p(y^{(i)} | \mathbf{x}^{(i)}; \theta)$$

- For linear regression with Gaussian noise, maximizing this term is mathematically identical to **minimizing the Mean Squared Error**.

  This provides a deep statistical justification for why we use least squares.

Logistic Regression

1. Loglikelihood:

$$l(\theta) = \ln L(\theta) = \sum_{i=1}^{m} \left[y^{(i)} \ln h_\theta(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \ln(1 - h_\theta(\mathbf{x}^{(i)})) \right]$$

2. Gradient Ascent (as we are maximizing the Loglikelihood):

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} l(\theta) &= \left(y \frac{1}{g(\theta^T \mathbf{x})} - (1-y) \frac{1}{1 - g(\theta^T \mathbf{x})} \right) \frac{\partial}{\partial \theta_j} g(\theta^T \mathbf{x}) \\
&= \left(y \frac{1}{g(\theta^T \mathbf{x})} - (1-y) \frac{1}{1 - g(\theta^T \mathbf{x})} \right) g(\theta^T \mathbf{x})(1 - g(\theta^T \mathbf{x})) \frac{\partial}{\partial \theta_j} (\theta^T \mathbf{x}) \\
&= (y(1 - g(\theta^T \mathbf{x})) - (1-y) g(\theta^T \mathbf{x})) x_j = (y - h_\theta(\mathbf{x})) x_j
\end{aligned}$$

$$\theta_{j|_{k+1}} = \theta_{j|_k} + \alpha (y - h_\theta(\mathbf{x})) x_j|_k$$

The Meaning of the Loss Function: Cross-Entropy

Log-Likelihood = Negative Cross-Entropy

The log-likelihood function for logistic regression is a specific instance of a general loss function called **Cross-Entropy**, which is fundamental to classification tasks.

The cost for a single data point is:

$$\text{Cost}(y, h_{\theta(x)}) = -[y \ln(h_{\theta(x)}) + (1 - y) \ln(1 - h_{\theta(x)})]$$

Where y is the true label (0 or 1) and $h_{\theta(x)}$ is the predicted probability.

If the true answer is 1 ($y = 1$):

- Cost $= -\ln(h_{\theta(x)})$
- The cost function heavily penalizes the model if it predicts a probability close to 0.

If the true answer is 0 ($y = 0$):

- Cost $= -\ln(1 - h_{\theta(x)})$
- The cost function heavily penalizes the model if it predicts a probability close to 1.

Why This Matters

This loss function ensures that the model is strongly penalized for being confident and wrong, driving the predicted probabilities toward the correct true values.

Interpretation of Linear Regression in MLE framework

1. Let us assume that the outputs and the inputs are related via

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

2. $\epsilon^{(i)}$ is an error term that captures either unmodeled effects, or random noise and are distributed IID (independently and identically distributed) according to a Gaussian distribution.

$$\text{So, } \epsilon^{(i)} \sim N(0, \sigma^2), \text{ that is, } p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

$$\text{Hence, } p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

3. Given X (which contains all the $x^{(i)}$'s) and θ, what is the distribution of the $y^{(i)}$? The probability of this is: $p(Y|X;\theta)$

Interpretation of Linear Regression in MLE framework

1. Likelihood function: explicit representation of this as a function of θ:

$$L(\theta) = L(\theta, X, Y) = p(Y|X;\theta) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)};\theta)$$

$$L(\theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

2. Maximum likelihood estimate of θ: Maximize L(θ)
3. Loglikelihood l(θ) = ln L(θ) - for simplicity and tractability

$$\text{MLE: } l(\theta) = \ln \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

$$= m \ln \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2$$

4. Maximizing likelihood is equivalent to minimizing the square of error. Least square regression is special case of MLE.

Classification Metrics - Confusion Matrix

The Foundation of Classification Metrics

For a binary classifier (e.g., "Fault" vs. "No Fault"), the Confusion Matrix summarizes performance by comparing predicted labels to the true labels.

		Predicted Class	
		Positive (1)	Negative (0)
Actual Class	Positive (1)	True Positive (TP)	False Negative (FN)
	Negative (0)	False Positive (FP)	True Negative (TN)

Classification Metrics - Confusion Matrix

The Foundation of Classification Metrics

For a binary classifier (e.g., "Fault" vs. "No Fault"), the Confusion Matrix summarizes performance by comparing predicted labels to the true labels.

		Predicted Class	
		Positive (1)	Negative (0)
Actual Class	Positive (1)	True Positive (TP)	False Negative (FN)
	Negative (0)	False Positive (FP)	True Negative (TN)

- **TP:** Correctly predicted positive.

Classification Metrics - Confusion Matrix

The Foundation of Classification Metrics

For a binary classifier (e.g., "Fault" vs. "No Fault"), the Confusion Matrix summarizes performance by comparing predicted labels to the true labels.

		Predicted Class	
		Positive (1)	Negative (0)
Actual Class	Positive (1)	True Positive (TP)	False Negative (FN)
	Negative (0)	False Positive (FP)	True Negative (TN)

- **TP:** Correctly predicted positive.
- **FN (Type II Error):** Incorrectly predicted negative. This is often the most costly error (e.g., missing a real fault).

Classification Metrics - Confusion Matrix

The Foundation of Classification Metrics

For a binary classifier (e.g., "Fault" vs. "No Fault"), the Confusion Matrix summarizes performance by comparing predicted labels to the true labels.

		Predicted Class	
		Positive (1)	Negative (0)
Actual Class	Positive (1)	True Positive (TP)	False Negative (FN)
	Negative (0)	False Positive (FP)	True Negative (TN)

- **TP:** Correctly predicted positive.
- **FN (Type II Error):** Incorrectly predicted negative. This is often the most costly error (e.g., missing a real fault).
- **FP (Type I Error):** Incorrectly predicted positive. (e.g., a false alarm).

Classification Metrics - Confusion Matrix

The Foundation of Classification Metrics

For a binary classifier (e.g., "Fault" vs. "No Fault"), the Confusion Matrix summarizes performance by comparing predicted labels to the true labels.

Actual Class		**Predicted Class** Positive (1)	Negative (0)
	Positive (1)	True Positive (TP)	False Negative (FN)
	Negative (0)	False Positive (FP)	True Negative (TN)

- **TP:** Correctly predicted positive.
- **FN (Type II Error):** Incorrectly predicted negative. This is often the most costly error (e.g., missing a real fault).
- **FP (Type I Error):** Incorrectly predicted positive. (e.g., a false alarm).
- **TN:** Correctly predicted negative.

Classification Metrics - Performance Measures

Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Overall correctness

Warning: Misleading with imbalanced classes

Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

Of predicted positives, how many are correct?

Recall (Sensitivity)

$$\text{Recall} = \frac{TP}{TP + FN}$$

Of actual positives, how many did we catch?

F1-Score

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Harmonic mean of Precision and Recall

ROC Curve and AUC

ROC Curve

Plots **True Positive Rate** vs. **False Positive Rate** across all classification thresholds.

$$\text{TPR} = \frac{TP}{TP + FN} = \text{Recall}$$

$$\text{FPR} = \frac{FP}{FP + TN}$$

AUC (Area Under Curve)

- AUC $= 1.0$: Perfect classifier
- AUC $= 0.5$: Random classifier
- AUC > 0.8: Good classifier

images/roc_curve_example.pn

Energy Application: Compare different algorithms for equipment failure prediction across various threshold settings.

Model Diagnostics and Validation

Regression Diagnostics

Residual Analysis:

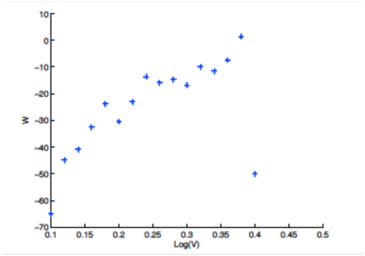
- **Residuals vs. Fitted:** Check for linearity and homoscedasticity
- **Q-Q Plot:** Check normality of residuals
- **Scale-Location:** Check for constant variance
- **Cook's Distance:** Identify influential points

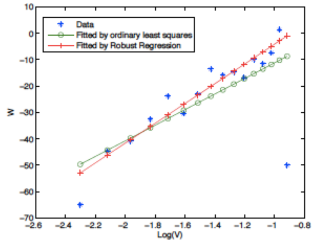
Classification Diagnostics

Performance Analysis:

- **Confusion Matrix:** Detailed error analysis
- **ROC/AUC:** Threshold-independent performance
- **Precision-Recall Curve:** Better for imbalanced classes
- **Calibration Plot:** Are predicted probabilities meaningful?

Robust Regression





1. Often the data has outliers. In such cases, the usual least square regression is dominated by these outliers which results in inaccurate model prediction.
2. To regulate the effect of such outlier points, the concept of robust regression (Huber's Regression) is used.

$$J(\theta) = \sum_{i=1}^{m} g(\epsilon^{(i)}) \quad ; \quad g(\epsilon) = \begin{cases} \frac{1}{2}\epsilon^2 & \text{if } |\epsilon| \le M \\ M\left(|\epsilon| - \frac{1}{2}M\right) & \text{if } |\epsilon| > M \end{cases}$$

ϵ is the error at each data point, and M is the threshold tolerance for outliers.

A New Challenge: When the Output is a Count

We moved from predicting continuous values to predicting the **probability of a class**. What about problems that are neither continuous nor binary?

- The number of accidents at an industrial site.
- The number of power outages in a day.
- The number of consumer complaints received per week.

Why our previous models don't fit:

- **Linear Regression:** Can predict negative values and doesn't account for the unique variance structure of count data.
- **Logistic Regression:** Only works for binary outcomes (0 or 1), not for counts that can be 2, 3, or more.

Poisson Regression: The Model for Counts

1. Poisson regression is designed specifically for modeling **count data**.
2. It assumes that the dependent variable Y follows a **Poisson distribution**.
3. The defining characteristic of the Poisson distribution is that its mean equals its variance ($E(Y) = \text{Var}(Y) = \lambda$).

The Poisson Probability Mass Function (PMF): $P(Y = k \mid \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$
λ: the expected count.

The Log Link Function

Similar to how logistic regression uses the sigmoid function to map to a probability, Poisson regression uses a **log link function** to connect the linear model to the expected count.

- We model the logarithm of the expected count, not the count itself.

$$\log(\lambda) = \beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k$$

- This is crucial because it ensures our predicted expected count $\lambda = e^{\text{linear model}}$ is **always positive**, which is a fundamental requirement for count data.

Interpreting the Coefficients

- Unlike linear regression (additive effects) or logistic regression (log-odds), the coefficients in Poisson regression have a **multiplicative interpretation**.
- A one-unit increase in a predictor variable X_j results in a change in the expected count by a factor of e^{β_j}.

Examples

- If $\beta_j = 0.5$:
$$e^{0.5} \approx 1.65 \quad \Rightarrow \quad \text{65\% increase in expected count}$$
- If $\beta_j = -0.2$:
$$e^{-0.2} \approx 0.82 \quad \Rightarrow \quad \text{18\% decrease in expected count}$$

Log-likelihood Function

Parameters β are estimated using **Maximum Likelihood Estimation (MLE)**.

For a Poisson-distributed variable Y:

$$L(\beta \mid X, Y) = \prod_{i=1}^{n} \frac{e^{-\lambda_i} \lambda_i^{y_i}}{y_i!}$$

where:

$$\lambda_i = e^{X_i \beta}, \quad y_i = \text{observed count for observation } i$$

$$\log L(\beta \mid X, Y) = \sum_{i=1}^{n} \big(y_i \log(\lambda_i) - \lambda_i - \log(y_i!) \big)$$

$$\log L(\beta \mid X, Y) = \sum_{i=1}^{n} \big(- e^{X_i \beta} + y_i X_i \beta - \log(y_i!) \big)$$

Maximization of log-likelihood Function

$$\frac{\partial}{\partial \beta_j}\left(-e^{X_i\beta}\right) = -\lambda_i X_{ij}$$

$$\frac{\partial}{\partial \beta_j}\left(y_i X_i \beta\right) = y_i X_{ij}$$

Combining:

$$\frac{\partial \log L(\beta)}{\partial \beta_j} = \sum_{i=1}^{n} (y_i - \lambda_i) X_{ij}$$

Gradient descent based update of parameters:

$$\beta^{(t+1)} = \beta^{(t)} + \alpha \sum_{i=1}^{n} X_i^T \left(y_i - \lambda_i\right)$$

where:

- $\beta^{(t)}$: parameter at iteration t
- α: learning rate
- X_i: predictor vector for obs. i
- y_i: observed count
- $\lambda_i = e^{X_i \beta^{(t)}}$: expected count

Applications in Energy Systems

Poisson regression models **count-based events** in energy systems:

- **Equipment Failures:** Predict breakdowns (transformers, turbines) from age, stress, temp., maintenance.
- **Service Interruptions:** Model outages $\sim$ weather, demand, infrastructure age.
- **Consumption Events:** Frequency of peak demand / high load conditions in smart grids.
- **Accidents:** Safety incidents (oil spills, fires) $\sim$ hours worked, equipment age, inspections.
- **Consumer Complaints:** Counts of outages, billing issues $\sim$ region, demographics, usage, season.

Interpretation and Advantages

Example: Outage Prediction
Predict outages using:

$$\log(\lambda) = \beta_0 + \beta_1 T + \beta_2 W + \beta_3 D + \beta_4 A$$

where $T =$ Temperature, $W =$ Wind speed, $D =$ Load demand, $A =$ Infrastructure age.
Interpretation:

- e^{β_j} = multiplicative effect on expected count.
- Example: $\beta_3 > 0 \Rightarrow$ higher demand $\rightarrow$ more outages.
- $\beta_4 > 0 \Rightarrow$ aging infrastructure $\rightarrow$ more failures.

Advantages:

- **Interpretability:** Direct link between predictors and event frequency.
- **Rare Events:** Handles failures, outages, accidents well.
- **Count Data:** Natural fit for discrete, non-negative outcomes.
- **Practical Impact:** Guides maintenance, planning, reliability, cost reduction.

Comparison of Regression Models in Energy Systems

Aspect	Linear	Logistic	Poisson
Dependent Var.	Continuous	Binary (0/1)	Counts (0,1,2,. . .)
Distribution	Normal $N(\mu, \sigma^2)$	Bernoulli p	Poisson λ
Link Function	Identity $Y = X\beta + \epsilon$	Logit $\log \frac{p}{1-p} = X\beta$	Log $\log(\lambda) = X\beta$
Coeff. Meaning	Change in mean	Change in log-odds	Change in log-count
Energy Example	Energy use	Failure prob.	Failure counts

Energy System Examples:

Linear: Energy Consumption $= \beta_0 + \beta_1(\text{Temp}) + \beta_2(\text{Number of Occupants}) + \epsilon$

Logistic: $\log\left(\frac{P(\text{Failure}=1)}{P(\text{Failure}=0)}\right) = \beta_0 + \beta_1(\text{Grid Load}) + \beta_2(\text{Temp})$

Poisson:

$log(\lambda) = \beta_0 + \beta_1(\text{Equipment Age}) + \beta_2(\text{Operating Temp}) + \beta_3(\text{Load Variability})$

Thank You