Linear Regression

Definition

A **supervised learning** method to model the relationship between a dependent variable (target) and one/more independent variables (features) by fitting a linear equation.

Mathematical Formulation

- **Hypothesis Function**:

$$h_w(\mathbf{X}) = w_0 + w_1 X_1 + w_2 X_2 + \cdots + w_n X_n$$

  - $\mathbf{w} = [w_0, w_1, \ldots, w_n]$: Weights (parameters)
  - $\mathbf{X} = [1, X_1, X_2, \ldots, X_n]$: Features (with bias term 1)

Cost Function (Mean Squared Error)

$$J(\mathbf{w}) = \frac{1}{2m} \sum_{i=1}^{m} (h_w(\mathbf{X}^{(i)}) - y^{(i)})^2$$

- m: Number of training examples
- Minimizing $J(\mathbf{w})$ gives optimal weights.

Solution Methods

1. Closed-Form Solution (Normal Equation)

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

- **Pros**: Exact solution, no iteration needed
- **Cons**: Slow for large datasets ($O(n^3)$ complexity)

2. Gradient Descent

Iterative weight update:

$$w_j := w_j - \alpha \frac{\partial J(\mathbf{w})}{\partial w_j}$$

For linear regression:

$$w_j := w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_w(\mathbf{X}^{(i)}) - y^{(i)}) X_j^{(i)}$$

- α: Learning rate
- Repeat until convergence.

Assumptions

1. **Linearity**: Relationship between features and target is linear.
2. **Independence**: Errors are uncorrelated.
3. **Homoscedasticity**: Constant error variance.
4. **Normality**: Errors follow normal distribution.

Evaluation Metrics

- **R-squared (Coefficient of Determination)**:

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

  - $$SS_{\text{res}} = \sum (y - \hat{y})^2$$
  - $$SS_{\text{tot}} = \sum (y - \bar{y})^2$$

- **Adjusted R-squared**:

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

Limitations

- Fails to capture nonlinear relationships.
- Sensitive to outliers.
- Assumes features are independent (no multicollinearity).

Linear Classification

Definition

A **supervised learning** method to predict discrete class labels by finding a linear decision boundary that separates classes in feature space.

Mathematical Formulation

- **Hypothesis Function** (for binary class $y \in 0, 1$):

$$h_w(\mathbf{X}) = \{ 1 \quad \text{if } \mathbf{w}^T \mathbf{X} \geq 0 \; 0 \quad \text{otherwise}$$

  - $\mathbf{w} = [w_0, w_1, \ldots, w_n]$: Weights (parameters)
  - $\mathbf{X} = [1, X_1, X_2, \ldots, X_n]$: Features with bias term
- **Decision Boundary**:

$$\mathbf{w}^T \mathbf{X} = 0$$

Solution Methods

1. Perceptron Algorithm

- **Update Rule** (for misclassified samples):

$$\mathbf{w} := \mathbf{w} + \alpha(y^{(i)} - \hat{y}^{(i)})\mathbf{X}^{(i)}$$

  - α: Learning rate
  - Iterates until all samples are correctly classified (if linearly separable).

2. Logistic Regression (Probabilistic Approach)

- **Hypothesis** (sigmoid function):

$$h_w(\mathbf{X}) = \sigma(\mathbf{w}^T\mathbf{X}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{X}}}$$

- **Cost Function** (Cross-Entropy Loss):

$$J(\mathbf{w}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_w(\mathbf{X}^{(i)})) + (1 - y^{(i)})\log(1 - h_w(\mathbf{X}^{(i)}))\right]$$

- **Gradient Descent Update**:

$$w_j := w_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_w(\mathbf{X}^{(i)}) - y^{(i)})X_j^{(i)}$$

Assumptions

1. **Linear Separability**: Classes can be separated by a hyperplane.
2. **Low Noise**: Minimal overlap between classes.
3. **Feature Independence**: Features are uncorrelated (ideal case).

Evaluation Metrics

- **Accuracy**: $\frac{\text{Correct Predictions}}{\text{Total Predictions}}$
- **Precision/Recall**: Trade-off between false positives/negatives.
- **F1-Score**: Harmonic mean of precision and recall.
- **ROC-AUC**: Area under the Receiver Operating Characteristic curve.

Limitations

- Fails for nonlinearly separable data.
- Sensitive to imbalanced datasets.
- Assumes equal importance of all features.

Linear Regression vs. Linear Classification

Feature	Linear Regression	Linear Classification
Problem Type	Regression (continuous output)	Classification (discrete labels)
Output	$y \in \mathbb{R}$	$y \in 0, 1$ (binary)
Hypothesis	$\mathbf{w}^T\mathbf{X}$	$\text{Sign}(\mathbf{w}^T\mathbf{X})$
Cost Function	Mean Squared Error (MSE)	Cross-Entropy Loss / Hinge Loss
Decision Boundary	Fits a line/plane to data	Separates classes with a hyperplane
Example Algorithms	Ordinary Least Squares	Perceptron, Logistic Regression

---

Key Differences

1. **Output**: Regression predicts continuous values; classification assigns discrete labels.
2. **Loss Function**: MSE (regression) vs. cross-entropy/hinge loss (classification).
3. **Evaluation**: R^2/RMSE (regression) vs. accuracy/precision (classification).
4. **Geometric Goal**: Minimize vertical distances (regression) vs. maximize margin (classification).

Logistic Regression (Logarithmic Regression)

Definition

A **probabilistic classification** method for binary outcomes ($y \in 0, 1$) that models the probability of a class using a logistic (sigmoid) function.
Key Idea: Transform linear regression output into a probability using the sigmoid.

Mathematical Formulation

- **Hypothesis Function**:

$$h_w(\mathbf{X}) = \sigma(\mathbf{w}^T\mathbf{X}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{X}}}$$

  - $\sigma(z)$: Sigmoid function squashes z to $[0, 1]$.

- **Interpretation**:
  $h_w(\mathbf{X}) = P(y = 1|\mathbf{X}; \mathbf{w})$

Cost Function (Cross-Entropy Loss)

$$J(\mathbf{w}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(h_w(\mathbf{X}^{(i)})) + (1 - y^{(i)})\log(1 - h_w(\mathbf{X}^{(i)}))\right]$$

Gradient Descent Update

$$w_j := w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_w(\mathbf{X}^{(i)}) - y^{(i)} \right) X_j^{(i)}$$

- α: Learning rate.

Assumptions

1. **Linearity**: Log-odds of the target are linear in features.
2. **Independent Observations**: No autocorrelation.
3. **No Multicollinearity**: Features are not highly correlated.

Multi-Class Classification

Definition

Predicting a discrete label from **>2 classes** (e.g., $y \in {0, 1, 2, \dots, K}$).

Solution Methods

1. One-vs-Rest (OvR)

- Train K binary classifiers, each distinguishing class k vs all others.
- **Prediction**: Class with highest probability.

2. One-vs-One (OvO)

- Train $\frac{K(K-1)}{2}$ classifiers for every pair of classes.
- **Prediction**: Majority vote across classifiers.

3. Softmax Regression (Multinomial Logistic Regression)

- Generalizes logistic regression to $K \geq 2$ classes.

Hypothesis Function

$$ P(y=k | \mathbf{X}; \mathbf{W}) = \frac{e^{\mathbf{w}k^T \mathbf{X}}}{\sum{i=1}^K e^{\mathbf{w}_i^T \mathbf{X}}} $$

- $\mathbf{W}$: Weight matrix ($K \times n$).
- Softmax normalizes outputs into probabilities summing to 1.

Cost Function (Cross-Entropy)

$$ J(\mathbf{W}) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K \mathbf{1}{y^{(i)} = k} \log\left(\frac{e^{\mathbf{w}k^T \mathbf{X}^{(i)}}}{\sum{j=1}^K e^{\mathbf{w}_j^T \mathbf{X}^{(i)}}} \right) $$

- $\mathbf{1}\{\dots\}$: Indicator function (1 if true, else 0).

Gradient Descent Update

$$ \mathbf{w}_k := \mathbf{w}*k* - \alpha *\frac{1}{m}* \sum*{i=1}^m \left[P(y=k|\mathbf{X}^{(i)}) - \mathbf{1}{y^{(i)} = k} \right] \mathbf{X}^{(i)} $$

Evaluation Metrics

- **Confusion Matrix**: Class-wise accuracy/errors.
- **Precision/Recall/F1-Score**: Extended to multi-class (macro/micro averaging).
- **Log-Loss**: Measures probabilistic calibration.

Key Challenges

- Class imbalance affects performance.
- Computational cost increases with K.
- Requires feature scaling for gradient-based methods.

Neuron Model & Perceptron

Biological Inspiration

- **Biological Neuron**:
  - Dendrites (input), Cell body (processing), Axon (output).
  - Communication via electrical impulses (spikes).
- **Artificial Neuron**: Simplified computational model inspired by biology.

Artificial Neuron (McCulloch-Pitts Model)

Definition

A mathematical unit that computes a weighted sum of inputs, applies an activation function, and produces an output. Basis for neural networks.

Mathematical Formulation

- **Inputs**: $\mathbf{X} = [x_0 = 1, x_1, x_2, \dots, x_n]$ (with bias $x_0 = 1$)
- **Weights**: $\mathbf{w} = [w_0, w_1, \dots, w_n]$
- **Pre-activation**:

$$z = \mathbf{w}^T\mathbf{X} = w_0 + \sum_{i=1}^{n} w_i x_i$$

- **Activation Function**:

$$y = \phi(z)$$

Common Activation Functions

Function	Formula	Use Case
Step	$\phi(z) = \begin{cases} 1 & z \geq 0 \\ 0 & \text{otherwise} \end{cases}$	Perceptron
Sigmoid	$\phi(z) = \frac{1}{1+e^{-z}}$	Logistic regression
ReLU	$\phi(z) = \max(0, z)$	Deep networks

Perceptron (Rosenblatt, 1957)

Definition

A **single-layer neural network** for binary linear classification. Learns weights to separate two classes using a step activation.

Algorithm

1. **Initialize Weights**: $\mathbf{w} = [0, 0, \dots, 0]$.
2. **For each training sample** $(\mathbf{X}^{(i)}, y^{(i)})$:
   - Compute prediction:

$$\hat{y}^{(i)} = \begin{cases} 1 & \mathbf{w}^T \mathbf{X}^{(i)} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

   - Update weights if misclassified:

$$\mathbf{w} := \mathbf{w} + \alpha (y^{(i)} - \hat{y}^{(i)}) \mathbf{X}^{(i)}$$

     - α: Learning rate (e.g., $\alpha = 0.1$).

Key Properties

- **Convergence**: Guaranteed if data is **linearly separable**.
- **Limitation**: Fails on nonlinearly separable data (e.g., XOR problem).

Perceptron vs. Logistic Regression Neuron

Feature	Perceptron	Logistic Regression Neuron
Activation	Step function	Sigmoid function
Output	Binary (0/1)	Probability (0-1)
Learning Rule	Direct weight updates on errors	Gradient descent on cross-entropy

Feature	Perceptron	Logistic Regression Neuron
Use Case	Hard classification	Probabilistic classification

Applications of Perceptron

- Simple binary classification (e.g., spam detection).
- Building block for multi-layer perceptrons (MLPs).

Limitations of Single Neurons/Perceptrons

1. Cannot solve nonlinear problems (e.g., XOR).
2. No probabilistic output (perceptron).
3. Requires manual feature engineering for complex tasks.

From Perceptron to Neural Networks

- Stack perceptrons into **hidden layers** to create MLPs.
- Use **nonlinear activation functions** (e.g., ReLU) to model complex patterns.
- Train with **backpropagation** and gradient descent.

Multi-Layer Perceptron (MLP)

Definition

A **feedforward neural network** with one or more hidden layers between the input and output layers. Capable of learning nonlinear decision boundaries.

Structure

- **Input Layer**: Raw features ($\mathbf{X}$).
- **Hidden Layer(s)**: Intermediate computations with nonlinear activations.
- **Output Layer**: Final prediction (e.g., class probabilities).

Mathematical Formulation (Forward Pass)

For layer l:

1. **Pre-activation**:

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}$$

   - $\mathbf{W}^{[l]}$: Weight matrix ($\text{units}^{[l]} \times \text{units}^{[l-1]}$)
   - $\mathbf{a}^{[l-1]}$: Activation from layer $l-1$ ($\mathbf{a}^{[0]} = \mathbf{X}$)

2. **Activation**:

$$\mathbf{a}^{[l]} = g^{[l]}(\mathbf{z}^{[l]})$$

   - $g^{[l]}$: Activation function (e.g., ReLU, sigmoid).

Common Activation Functions

Function	Formula	Derivative
Sigmoid	$g(z) = \frac{1}{1+e^{-z}}$	$g'(z) = g(z)(1 - g(z))$
ReLU	$g(z) = \max(0, z)$	$g'(z) = \{1 \quad z > 0 \; 0 \quad \text{otherwise}$
Tanh	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g'(z) = 1 - g(z)^2$

Backpropagation (BP) Algorithm

Definition

An algorithm to compute gradients of the loss with respect to weights by applying the **chain rule** backward from output to input.

Steps

1. **Forward Pass**: Compute predictions and loss (e.g., MSE, cross-entropy).
2. **Backward Pass**:
   - Compute error gradient at output layer.
   - Propagate error backward to update weights in hidden layers.

Key Equations

Loss Functions

- **MSE**:

$$J = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

- **Cross-Entropy**:

$$J = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

Gradient Computation

For layer l (starting from output layer L):

1. **Output Layer Error**:

$$\delta^{[L]} = \frac{\partial J}{\partial \mathbf{z}^{[L]}} = (\mathbf{a}^{[L]} - \mathbf{y}) \odot g'^{[L]}(\mathbf{z}^{[L]})$$

   - $\odot$: Element-wise multiplication.

2. **Hidden Layer Error**:

$$\delta^{[l]} = (\mathbf{W}^{[l+1]T}\delta^{[l+1]}) \odot g'^{[l]}(\mathbf{z}^{[l]})$$

3. **Weight Gradients**:

$$\frac{\partial J}{\partial \mathbf{W}^{[l]}} = \frac{1}{m}\delta^{[l]}\mathbf{a}^{[l-1]T}$$

$$\frac{\partial J}{\partial \mathbf{b}^{[l]}} = \frac{1}{m}\sum_{i=1}^{m}\delta^{[l]}$$

Weight Update (Gradient Descent)

$$\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \alpha\frac{\partial J}{\partial \mathbf{W}^{[l]}}$$

$$\mathbf{b}^{[l]} := \mathbf{b}^{[l]} - \alpha\frac{\partial J}{\partial \mathbf{b}^{[l]}}$$

- α: Learning rate.

Training Challenges

1. **Vanishing/Exploding Gradients**: Mitigated with ReLU, batch normalization.
2. **Overfitting**: Addressed by dropout, L2 regularization.
3. **Local Minima**: Stochastic gradient descent helps escape.

MLP vs. Single-Layer Perceptron

Feature	MLP	Single-Layer Perceptron
Layers	≥1 hidden layers	No hidden layers
Nonlinearity	Learns nonlinear patterns	Linear decision boundary
Training	Backpropagation + gradient descent	Perceptron update rule
Use Cases	Complex tasks (e.g., image recognition)	Linearly separable problems

---

Initialization & Regularization

- **Weight Initialization**: Xavier/He initialization.
- **Regularization**:
  - L2: $J = J + \frac{\lambda}{2m} \sum |\mathbf{W}|^2$
  - Dropout: Randomly deactivate neurons during training.