Week 1: Introduction to Machine Learning and Linear Regression

Table of Contents

---

1. Introduction to Machine Learning {#introduction}

What is Machine Learning?

Machine Learning is a field of study that gives computers the ability to learn without being explicitly programmed. It's about creating algorithms that can learn patterns from data and make predictions or decisions.

Applications of Machine Learning

- **Web Search**: Ranking web pages, personalized search results
- **Photo Tagging**: Automatically identifying people in photos
- **Email Spam Filtering**: Detecting and filtering unwanted emails
- **Recommendation Systems**: Netflix, Amazon, Spotify recommendations
- **Medical Diagnosis**: Analyzing medical images, drug discovery
- **Autonomous Vehicles**: Self-driving cars, route optimization
- **Financial Services**: Credit scoring, fraud detection
- **Natural Language Processing**: Language translation, chatbots

Why Machine Learning Matters

- Handles complex patterns in large datasets
- Adapts to new data automatically
- Scales efficiently with increasing data
- Powers modern AI applications

---

2. Types of Machine Learning {#types}

Supervised Learning

Learning from labeled training data to make predictions on new, unseen data.

Characteristics:

- Uses input-output pairs (x, y)
- Algorithm learns mapping from input to output
- Performance can be measured against known correct answers

Types of Supervised Learning:

1. Regression

- Predicts continuous numerical values
- Output is a number from infinite set of possible values
- Examples:
  - House price prediction
  - Stock price forecasting
  - Temperature prediction

2. Classification

- Predicts discrete categories or classes
- Output is from finite set of possible values
- Examples:
  - Email spam detection (spam/not spam)
  - Medical diagnosis (disease/no disease)
  - Image recognition (cat/dog/bird)

Unsupervised Learning

Finding patterns in data without labeled examples.

Types of Unsupervised Learning:

1. Clustering

- Groups similar data points together

- Examples:
  - Customer segmentation
  - Gene sequencing
  - Market research

2. Anomaly Detection

- Identifies unusual patterns
- Examples:
  - Fraud detection
  - System monitoring
  - Quality control

3. Dimensionality Reduction

- Reduces number of features while preserving important information
- Examples:
  - Data visualization
  - Feature extraction
  - Noise reduction

---

3. Linear Regression {#linear-regression}

Model Representation

Linear regression models the relationship between input features and output using a linear equation.

Mathematical Notation:

- `x` = input variable (feature)
- `y` = output variable (target)
- `m` = number of training examples
- `(x^(i), y^(i))` = i-th training example

Linear Regression Formula

For single feature (univariate):

```
f(x) = wx + b
```

Where:

- `w` = weight (slope)
- `b` = bias (y-intercept)
- `f(x)` = predicted output

Model Components

1. **Training Set**: Historical data used to train the model
2. **Learning Algorithm**: Process that finds optimal parameters
3. **Hypothesis/Model**: Function that makes predictions
4. **Parameters**: Values (w, b) that define the model

Example: House Price Prediction

```python
# Simple linear regression example
import numpy as np
import matplotlib.pyplot as plt

# Training data
x_train = np.array([1, 2, 3, 4, 5])  # House size (1000 sq ft)
y_train = np.array([1, 2, 3, 4, 5])  # Price (100k $)

# Model parameters
w = 1.0  # weight
b = 0.0  # bias

# Prediction function
def predict(x):
    return w * x + b

# Make predictions
predictions = predict(x_train)
print(f"Predictions: {predictions}")
```

---

4. Cost Function {#cost-function}

Purpose

The cost function measures how well the model fits the training data by calculating the difference between predicted and actual values.

Squared Error Cost Function

Most common cost function for linear regression:

```
J(w,b) = (1/2m) * Σ(f(x^(i)) - y^(i))²
```

Where:

- `J(w,b)` = cost function
- `m` = number of training examples
- `f(x^(i))` = predicted value for i-th example
- `y^(i)` = actual value for i-th example

Why Squared Error?

- Penalizes larger errors more heavily
- Mathematically convenient (differentiable)
- Leads to unique global minimum
- Widely used and well-understood

Cost Function Intuition

- **Goal**: Minimize J(w,b) to find best fit line
- **Process**: Try different values of w and b
- **Result**: Parameters that minimize average squared error

Implementation Example

```python
```

```
def compute_cost(x, y, w, b):
    m = len(x)
    cost = 0

    for i in range(m):
        prediction = w * x[i] + b
        cost += (prediction - y[i]) ** 2

    return cost / (2 * m)

# Example calculation
x = np.array([1, 2, 3])
y = np.array([1, 2, 3])
w, b = 1.0, 0.0

cost = compute_cost(x, y, w, b)
print(f"Cost: {cost}")
```

Visualization

The cost function forms a bowl-shaped curve (convex function) when plotted against parameters, with a single global minimum.

---

5. Gradient Descent {#gradient-descent}

Overview

Gradient descent is an optimization algorithm used to minimize the cost function by iteratively adjusting parameters.

Algorithm Steps

1. Start with initial parameter values
2. Calculate gradient of cost function
3. Update parameters in direction of steepest descent
4. Repeat until convergence

Mathematical Formulation

Simultaneous Update Rules:

```
w = w - α * ∂J(w,b)/∂w
b = b - α * ∂J(w,b)/∂b
```

Where:

- `α` = learning rate
- `∂J(w,b)/∂w` = partial derivative of cost with respect to w
- `∂J(w,b)/∂b` = partial derivative of cost with respect to b

Partial Derivatives for Linear Regression

```
∂J(w,b)/∂w = (1/m) * Σ(f(x^(i)) - y^(i)) * x^(i)
∂J(w,b)/∂b = (1/m) * Σ(f(x^(i)) - y^(i))
```

Learning Rate (α)

Critical hyperparameter that controls step size:

- **Too small**: Slow convergence, many iterations needed
- **Too large**: May overshoot minimum, fail to converge
- **Just right**: Efficient convergence to minimum

Gradient Descent Intuition

1. **Derivative > 0**: Function increasing, move left (decrease parameter)
2. **Derivative < 0**: Function decreasing, move right (increase parameter)
3. **Derivative = 0**: At minimum, no update needed

Implementation Example

```python
```

```python
def gradient_descent(x, y, w_init, b_init, alpha, num_iterations):
    w = w_init
    b = b_init
    m = len(x)

    for i in range(num_iterations):
        # Calculate predictions
        predictions = w * x + b
        
        # Calculate gradients
        dw = (1/m) * np.sum((predictions - y) * x)
        db = (1/m) * np.sum(predictions - y)
        
        # Update parameters
        w = w - alpha * dw
        b = b - alpha * db
    
    return w, b

# Example usage
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])
w, b = gradient_descent(x, y, 0.0, 0.0, 0.01, 1000)
```

Convergence

- Algorithm converges when gradient approaches zero
- Cost function decreases with each iteration
- Parameters stabilize at optimal values

---

6. Implementation Examples {#implementation}

Complete Linear Regression Implementation

```
python
```

```python
import numpy as np
import matplotlib.pyplot as plt

class LinearRegression:
    def __init__(self, learning_rate=0.01, max_iterations=1000):
        self.learning_rate = learning_rate
        self.max_iterations = max_iterations
        self.w = 0
        self.b = 0
        self.cost_history = []

    def fit(self, X, y):
        m = len(X)

        for i in range(self.max_iterations):
            # Forward pass
            predictions = self.w * X + self.b

            # Compute cost
            cost = np.sum((predictions - y) ** 2) / (2 * m)
            self.cost_history.append(cost)

            # Compute gradients
            dw = np.sum((predictions - y) * X) / m
            db = np.sum(predictions - y) / m

            # Update parameters
            self.w -= self.learning_rate * dw
            self.b -= self.learning_rate * db

    def predict(self, X):
        return self.w * X + self.b

# Example usage
X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

model = LinearRegression(learning_rate=0.01, max_iterations=1000)
model.fit(X, y)

print(f"Learned parameters: w = {model.w:.2f}, b = {model.b:.2f}")
print(f"Predictions: {model.predict(X)}")
```

Vectorized Implementation

```python
def vectorized_gradient_descent(X, y, alpha=0.01, iterations=1000):
    m = len(X)
    w = 0
    b = 0
    
    for i in range(iterations):
        # Vectorized prediction
        predictions = w * X + b
        
        # Vectorized gradient computation
        dw = (1/m) * np.dot((predictions - y), X)
        db = (1/m) * np.sum(predictions - y)
        
        # Update parameters
        w -= alpha * dw
        b -= alpha * db
    
    return w, b

# Usage
X = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])
w, b = vectorized_gradient_descent(X, y)
```

Key Takeaways

1. **Machine Learning Types**: Supervised (regression/classification) and unsupervised (clustering/anomaly detection)
2. **Linear Regression**: Models linear relationships using `f(x) = wx + b`
3. **Cost Function**: Measures model performance using squared error
4. **Gradient Descent**: Optimization algorithm that minimizes cost by updating parameters iteratively
5. **Learning Rate**: Critical hyperparameter that controls convergence speed
6. **Implementation**: Can be done using loops or vectorized operations for efficiency

---

Next Steps

- Week 2: Multiple linear regression with multiple features
- Advanced optimization techniques
- Feature scaling and normalization
- Polynomial regression and regularization