Week 3: Introduction to Classification and Logistic Regression

1. Introduction to Classification

- Classification is a supervised learning task where the output variable y takes on a small, finite number of possible values.
- Unlike linear regression, which predicts continuous numbers, classification predicts categories.

Examples of Classification Problems

Problem	Possible Outputs	Type
Spam Detection	Spam (Yes) / Not Spam (No)	Binary Classification
Fraud Detection	Fraudulent / Legitimate	Binary Classification
Tumor Classification	Malignant / Benign	Binary Classification

Binary Classification

- The output label y can be one of two possible values: 0 or 1.
- **Common Terminology:**
  - **Negative class (0):** Absence of a feature (e.g., not spam, benign tumor).
  - **Positive class (1):** Presence of a feature (e.g., spam, malignant tumor).
  - **Interchangeable labels:** No/Yes, False/True, 0/1.

2. Why Linear Regression Fails for Classification

- Linear regression predicts values beyond 0 and 1, which is problematic for classification.
- **Thresholding technique:** If $f(x) < 0.5$, classify as 0; if $f(x) \geq 0.5$, classify as 1.
- **Problem:** Adding a single extreme data point can shift the decision boundary significantly, leading to incorrect classifications.

3. Introducing Logistic Regression

- **Logistic Regression is designed for binary classification.**
- Unlike linear regression, logistic regression ensures outputs remain between 0 and 1.
- The algorithm models the probability that $y = 1$ given x.
- **Misleading name:** Despite having "regression" in its name, it is used for classification.

4. Optional Lab

- Experiment with using linear regression for classification.
- Observe how it often fails and motivates the need for logistic regression.

Next Steps

- Learn about **decision boundaries** in the next section.
- Explore the logistic function and its mathematical formulation.

Logistic Regression - Comprehensive Notes

1. Introduction to Logistic Regression

- Logistic Regression is one of the most widely used classification algorithms.
- Unlike linear regression, which predicts continuous values, logistic regression predicts probabilities for classification.
- Example: **Tumor Classification**
  - **1 (Yes)** → Malignant Tumor (Positive Class)
  - **0 (No)** → Benign Tumor (Negative Class)
  - The goal is to classify tumors based on their size.

2. Why Not Use Linear Regression?

- Linear regression would try to fit a straight line, which can produce values beyond 0 and 1.
- Classification requires a function that maps inputs to a probability range **[0,1]**.
- Logistic Regression solves this by using the **Sigmoid Function**.

3. Sigmoid (Logistic) Function

- **Definition**: A function that maps any real-valued number to a range between **0 and 1**.
- **Formula**:

$$g(z) = \frac{1}{1 + e^{-z}}$$

  - e is Euler's number (~2.718).
  - z is the input (a linear combination of weights and features).
- **Properties**:
  - When z is **large**, e^{-z} is very small → $g(z)$ approaches **1**.
  - When z is **negative**, e^{-z} is large → $g(z)$ approaches **0**.
  - When $z = 0$, $g(0) = 0.5$.

4. Logistic Regression Model

- **Step 1**: Compute **z** (Linear Function)

$$z = w \cdot x + b$$

- **w**: Weights (parameters)
- **x**: Features (input data)
- **b**: Bias term

- **Step 2**: Apply the Sigmoid Function

$$f(x) = g(z) = \frac{1}{1 + e^{-z}}$$

- **Interpretation**:
  - The output is the **probability** that $y = 1$ given x.
  - Example: If $f(x) = 0.7$, there is a **70% chance** that the tumor is malignant.

5. Probability Interpretation

- $P(y = 1|x)$ is the probability of class **1**.
- $P(y = 0|x) = 1 - P(y = 1|x)$.
- If $P(y = 1) = 0.7$, then $P(y = 0) = 0.3$.
- **Notation**:

$$f(x) = P(y = 1|x; w, b)$$

  - w, b are parameters learned during training.

6. Decision Boundary

- **Threshold-Based Prediction**:
  - If $f(x) \geq 0.5$, predict **y = 1**.
  - If $f(x) < 0.5$, predict **y = 0**.
- **Mathematical Interpretation**:
  - Since $f(x) = g(z)$,
  - $g(z) \geq 0.5$ when $z \geq 0$ (since $g(0) = 0.5$).
  - Hence, **decision boundary** occurs when:

$$w \cdot x + b = 0$$

- **Visualization**:
  - **For one feature**: A single threshold on a number line.
  - **For two features**: A straight line in a 2D space.
  - **For multiple features**: A hyperplane in n-dimensional space.

7. Example: Decision Boundary for Two Features

- Given two features x_1, x_2:

$$z = w_1 x_1 + w_2 x_2 + b$$

- Suppose:
  - $w_1 = 1, w_2 = 1, b = -3.$
  - Decision boundary is:

$$x_1 + x_2 = 3$$

  - Region where $x_1 + x_2 \geq 3 \rightarrow$ Predict **1**.
  - Region where $x_1 + x_2 < 3 \rightarrow$ Predict **0**.

8. Non-Linear Decision Boundaries

- **Adding Polynomial Features**:
  - Example: Using quadratic terms

$$z = w_1 x_1^2 + w_2 x_2^2 + b$$

  - This results in a **circular decision boundary**.
  - More complex polynomials → More complex decision boundaries.
- **Example**:
  - $x_1^2 + x_2^2 = 1$ (a circle).
  - Points inside the circle → Predict **0**.
  - Points outside the circle → Predict **1**.

9. Key Takeaways

Concept	Explanation
Logistic Regression	A classification algorithm that predicts probabilities.
Sigmoid Function	Maps input values to probabilities between **0 and 1**.
Probability Interpretation	$f(x)$ gives the probability of class **1**.
Decision Boundary	A threshold where the model transitions from class 0 to class 1.
Linear Decision Boundaries	Logistic Regression naturally produces **linear** decision boundaries.
Non-Linear Boundaries	Can be achieved using **polynomial features**.

10. Next Steps

- Understanding the **cost function** for logistic regression.

- Applying **gradient descent** for training the model.
- Implementing **logistic regression in code**.

This note covers all major points from the lecture with structured explanations. Let me know if you want any modifications! 🚀

Logistic Regression - Cost Function and Loss Function

1. Introduction

- The **cost function** measures how well a set of parameters fits the training data.
- The goal is to choose parameters **w** and **b** to minimize the cost function.
- In **linear regression**, we used the **squared error cost function**, but it doesn't work well for logistic regression.

---

2. Why Squared Error is Not Suitable for Logistic Regression?

Reason	Explanation
Non-convexity	The squared error cost function produces a **wiggly** cost surface with **multiple local minima**, making it difficult for gradient descent to converge to a global minimum.
Gradient Issues	The sigmoid function in logistic regression causes the derivative of squared error to be **very small** for large or small values, slowing down learning.

---

3. Loss Function for Logistic Regression

- Instead of using **squared error**, we define a new **log loss function**.
- The **loss function** measures the error on a **single training example**.
- Given **prediction** $f(x)$ and **true label** y, the loss function is:

$$L(f(x), y) = \begin{cases} -\log(f(x)), & \text{if } y = 1 \\ -\log(1 - f(x)), & \text{if } y = 0 \end{cases}$$

Why Log Loss?

1. **For $y = 1$:**
   - If $f(x) \approx 1$ (correct prediction), **loss is small**.
   - If $f(x) \approx 0$ (wrong prediction), **loss is large**.
2. **For $y = 0$:**
   - If $f(x) \approx 0$ (correct prediction), **loss is small**.
   - If $f(x) \approx 1$ (wrong prediction), **loss is large**.

Key property: The function penalizes wrong predictions more, ensuring better convergence.

---

4. Cost Function for Logistic Regression

- The **cost function** is the average loss over **all training examples**.
- Given m training examples, the cost function is:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)}), y^{(i)})$$

- Expanding using log loss:

$$J(w,b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(f(x^{(i)})) + (1-y^{(i)})\log(1-f(x^{(i)}))\right]$$

---

5. Convexity of the Cost Function

- Unlike squared error, this function is **convex**, meaning it has a single **global minimum**.
- **Gradient Descent** can now reliably optimize w and b.

---

6. Summary

Concept	Explanation
Loss Function	Measures the error for a single training example. Uses log loss to ensure proper penalty.
Cost Function	Average loss over all training examples. Ensures smooth optimization.
Why Not Squared Error?	Creates multiple local minima, making gradient descent ineffective.
Why Log Loss?	Ensures correct penalty for wrong predictions and smooth convergence.
Convexity	The new cost function is convex, making optimization easier.

---

7. Next Steps

- Use **gradient descent** to optimize w and b.
- Implement **logistic regression** in code.

This note covers all key ideas logically. Let me know if you need refinements! 🚀

Vectorization in Machine Learning

1. Introduction to Vectorization

Vectorization is a powerful technique used in machine learning that makes code **shorter** and **more efficient**. It allows computations to leverage **modern numerical linear algebra libraries** and even **GPU hardware** for faster execution.

Benefits of Vectorization

Benefit	Explanation
Concise Code	Reduces the number of lines of code, making it easier to read and write.
Improved Efficiency	Utilizes parallel computing to speed up execution.
Utilizes Modern Hardware	Leverages optimized libraries like **NumPy** and hardware like **GPUs**.

2. Example of Vectorization

Consider a machine learning model with parameters:

- w (vector of weights)
- x (vector of features)
- b (bias term)

For a dataset with **n = 3** features:

- $w = [w_1, w_2, w_3]$
- $x = [x_1, x_2, x_3]$

Without Vectorization (Using a For Loop)

Using a loop to compute the linear function:

$$f = \sum_{j=1}^{n} w_j \cdot x_j + b$$

- **Python Code (Inefficient)**:

```python
f = 0
for j in range(n):
    f += w[j] * x[j]
f += b
```

- **Problems**:
  - Inefficient when **n is large** (e.g., 100,000).
  - Computationally expensive due to **sequential execution**.

With Vectorization (Using NumPy)

- **Mathematical Expression**:

$$f = w \cdot x + b$$

- **Python Code (Efficient)**:

```python
f = np.dot(w, x) + b
```

- **Advantages**:
  - **Single-line implementation**.
  - **Uses parallel computing**, making it significantly **faster**.

---

3. Why is Vectorization Faster?

Sequential Computation (Without Vectorization)

- Executes operations **one after another**.
- For **large datasets**, it is **slow**.

Parallel Computation (With Vectorization)

- Uses specialized **hardware (CPU/GPU)** to process **multiple values at the same time**.
- Takes advantage of optimized **NumPy** functions.

Approach	Execution Style	Speed
For Loop	Sequential execution	Slow
Vectorized NumPy	Parallel execution	Fast

Example: Computing a Dot Product

Without Vectorization:

Each multiplication happens one by one:

1. $w_1 \times x_1$
2. $w_2 \times x_2$
3. $w_3 \times x_3$
   ...until $w_n \times x_n$

With Vectorization:

- The computer processes **all multiplications simultaneously** using **parallel execution**.
- Uses **specialized hardware (SIMD, GPU, CPU vector instructions)**.

---

4. Application in Machine Learning

Gradient Descent Update (Without Vectorization)

For multiple parameters:

$$w_j = w_j - \alpha \cdot d_j$$

Where:

- α = learning rate
- d_j = gradient of w_j

Python Code (Without Vectorization):

```python
for j in range(n):
    w[j] = w[j] - 0.1 * d[j]
```

Problem: Slow when updating **thousands of parameters**.

Gradient Descent Update (With Vectorization)

Mathematical Formula:

$$w = w - \alpha \cdot d$$

Python Code (Vectorized Implementation):

```python
w = w - 0.1 * d
```

- **Computes all updates in parallel**.
- **Reduces computation time significantly**.

---

5. Summary

Aspect	Without Vectorization	With Vectorization
Implementation	Uses loops	Uses matrix operations
Execution	Sequential	Parallel
Speed	Slow	Fast
Code Complexity	Long	Short

Key Takeaways

✅ Vectorization reduces code complexity.
✅ Improves performance by utilizing parallel computation.
✅ Essential for **large-scale machine learning models**.

Vectorization is a **fundamental technique** in **machine learning**, enabling algorithms to handle **large datasets** efficiently. 🚀

Gradient Descent for Multiple Linear Regression with Vectorization

1. Introduction

You have learned about **gradient descent, multiple linear regression, and vectorization**. Now, let's combine these concepts to implement gradient descent for multiple linear regression **efficiently**.

Why Vectorization?

- It **makes code shorter** and more readable.
- It **runs much faster** due to parallel computations.
- It **utilizes modern hardware** (like GPUs).

---

2. Multiple Linear Regression in Vector Notation

2.1 Traditional Representation

A multiple linear regression model predicts a target y using multiple features $x_1, x_2, \dots, x_n$:

$$f(x) = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b$$

where:

- $w_1, w_2, \dots, w_n$ are parameters (weights).
- $x_1, x_2, \dots, x_n$ are features.
- b is the bias term.

2.2 Vectorized Representation

Instead of treating $w_1, w_2, \dots, w_n$ as separate numbers, we **collect them into a vector**:

$$f(x) = \mathbf{w}^T \mathbf{x} + b$$

where:

- $\mathbf{w}$ (weight vector) = $[w_1, w_2, \dots, w_n]$.
- $\mathbf{x}$ (feature vector) = $[x_1, x_2, \dots, x_n]$.

- $\mathbf{w}^T\mathbf{x}$ is the **dot product** of $\mathbf{w}$ and $\mathbf{x}$.

In **NumPy**, this can be implemented efficiently as:

```
python

f_x = np.dot(w, x) + b
```

---

3. Cost Function for Multiple Linear Regression

3.1 Traditional Representation

The cost function $J(w, b)$ measures how well the model fits the data:

$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} \left(f(x^{(i)}) - y^{(i)} \right)^2$$

where:

- m is the number of training examples.
- $f(x^{(i)})$ is the model's prediction for the ith example.
- $y^{(i)}$ is the actual target value.

3.2 Vectorized Representation

Using matrix operations, we can write:

$$J(w,b) = \frac{1}{2m} ||Xw + b - Y||^2$$

where:

- X is an $m \times n$ matrix of feature values.
- Y is an $m \times 1$ vector of target values.

In **NumPy**, this can be computed efficiently as:

```
python

J = (1 / (2 * m)) * np.sum((np.dot(X, w) + b - Y) ** 2)
```

---

4. Gradient Descent for Multiple Linear Regression

4.1 Traditional Update Rules

Gradient descent updates each parameter using:

$$w_j := w_j - \alpha \frac{\partial J}{\partial w_j}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

where:

- α is the learning rate.
- $\frac{\partial J}{\partial w_j}$ and $\frac{\partial J}{\partial b}$ are partial derivatives of the cost function.

4.2 Vectorized Update Rule

Instead of updating each w_j individually, we update the entire weight vector at once:

$$\mathbf{w} := \mathbf{w} - \alpha \frac{1}{m} X^T (X\mathbf{w} + b - Y)$$

$$b := b - \alpha \frac{1}{m} \sum (X\mathbf{w} + b - Y)$$

In **NumPy**, this can be efficiently implemented as:

```python
grad_w = (1 / m) * np.dot(X.T, (np.dot(X, w) + b - Y))
grad_b = (1 / m) * np.sum(np.dot(X, w) + b - Y)

w -= alpha * grad_w
b -= alpha * grad_b
```

Why is this Efficient?

- Uses **parallel computations** instead of looping over each feature.
- Runs on **optimized linear algebra libraries (like BLAS and LAPACK)**.
- Can be **accelerated using GPUs**.

---

5. The Normal Equation (Alternative to Gradient Descent)

Instead of using an iterative process like gradient descent, the **normal equation** provides a direct solution:

$$w = (X^T X)^{-1} X^T Y$$

Advantages

✅ **No need for tuning a learning rate** (α).
✅ **Finds the optimal solution in one step** (for linear regression only).

Disadvantages

❌ **Does not generalize** to other ML algorithms (e.g., logistic regression, neural networks).
❌ **Computationally expensive** for large datasets (due to matrix inversion).

Most **ML libraries** (e.g., `` `scikit-learn` ``) use this method behind the scenes when fitting linear regression models.

---

6. Summary

Concept	Explanation
Multiple Linear Regression	Predicts y using multiple features $x_1, x_2, \ldots, x_n$.
Vectorization	Uses matrix operations to make computations faster.
Gradient Descent	Updates parameters iteratively to minimize cost function.
Vectorized Gradient Descent	Updates all parameters at once using matrix operations.
Normal Equation	Directly solves for w without iterations (only for linear regression).

---

7. What's Next?

- Learn how to **scale features** for better gradient descent convergence.
- Choose an **optimal learning rate (α)** to improve performance.
- Apply these concepts to **real-world datasets**.

Next up: Techniques to optimize multiple linear regression. 🚀