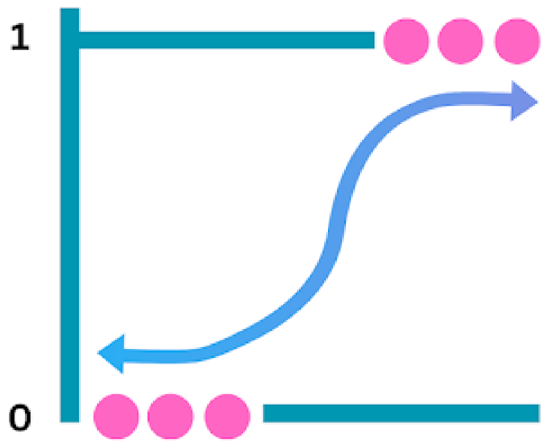
Classification in Logistic Regression

Logistic Regression

Logistic Regression is a statistical method used for binary classification problems, where the outcome can be one of two possible classes. Despite its name, it is a classification algorithm, not a regression algorithm.



Key Concepts:

Binary Classification:

Logistic regression is used when the dependent variable is binary (e.g., yes/no, true/false, 0/1). For binary classification, the default threshold of a logistic regression model is 0.5, which means that data points with a higher probability than 0.5 will automatically be assigned a label of 1. This threshold value can be manually changed depending on our use case to achieve better results.

Logistic Function (Sigmoid Function):

- The logistic function is used to model the probability of the default class (e.g., class 1). The function is defined as:

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

- This function maps any real-valued number into a value between 0 and 1, which can be interpreted as a probability.

Odds and Log-Odds:

Odds:

Odds represent the ratio of the probability of an event occurring to the probability of it not occurring. If p is the probability of an event occurring, the odds are calculated as:

$$\text{Odds} = \frac{p}{1-p}$$

For example, if the probability of success is 0.8, the odds of success are 0.8 / 1−0.8 = 4. This means the event is 4 times more likely to occur than not.

Log-Odds (Logit):

Log-odds is the natural logarithm of the odds:

$$\text{Logit}(p) = \log\left(\frac{p}{1-p}\right)$$

Log-odds can take any real value, from −∞ to +∞. When p=0.5, the log-odds is 0; when p>0.5, the log-odds is positive; and when p<0.5, the log-odds is negative.

Logistic Regression Model:

- The model predicts the log-odds of the probability of the default class as a linear combination of the input features.

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n$$

- This can be rearranged to give the logistic regression equation:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n)}}$$

Cost Function:

The cost function (or loss function) in logistic regression is typically the **log-loss function**, also known as **cross-entropy loss**. It measures how well the model's predicted probabilities match the actual class labels.

Cost Function for Single Variable Logistic Regression

The cost function J(m,c) for logistic regression with a single feature is:

$$J(m,c) = -\frac{1}{m} \sum_{i=1}^{m} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

where:

- m is the number of training examples.
- y_i is the actual label for the i-th training example.
- p_i is the predicted probability for the i-th training example, given by:

  $p_i = \frac{1}{1+e^{-(mx_i+c)}}$

Gradient Descent for Single Variable Logistic Regression

To find the optimal values of m and c, we use gradient descent. Here's how you can update m and c:

1. Initialize Parameters: Start with initial guesses for m and c, such as zeros or small random values.

2. Compute Cost: Calculate the cost J(m,c) using the formula above with the current values of m and c.

3. Compute Gradients: The gradient of the cost function with respect to **m** and **c** is:

$$\frac{\partial J(m,c)}{\partial m} = \frac{1}{m}\sum_{i=1}^{m}(p_i - y_i)\,x_i$$

$$\frac{\partial J(m,c)}{\partial c} = \frac{1}{m}\sum_{i=1}^{m}(p_i - y_i)$$

4. Update Parameters: Adjust the parameters m and c using the gradient descent update rules:

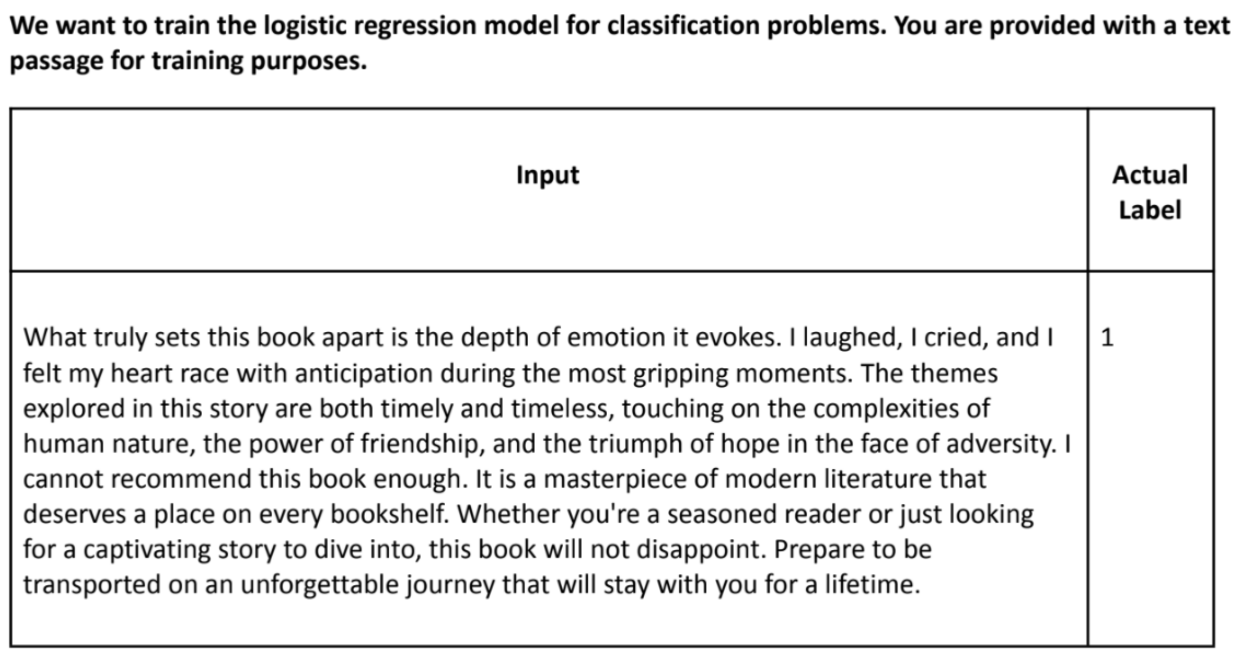
$$m := m - \alpha\frac{\partial J(m,c)}{\partial m}$$

$$c := c - \alpha\frac{\partial J(m,c)}{\partial c}$$

where α is the learning rate.

5. Repeat: Perform steps 2-4 iteratively until the cost function J(m,c) converges or a predetermined number of iterations is reached.

Text Sentiments Classification



We want to train the logistic regression model for classification problems. You are provided with a text passage for training purposes.

Input	**Actual Label**
What truly sets this book apart is the depth of emotion it evokes. I laughed, I cried, and I felt my heart race with anticipation during the most gripping moments. The themes explored in this story are both timely and timeless, touching on the complexities of human nature, the power of friendship, and the triumph of hope in the face of adversity. I cannot recommend this book enough. It is a masterpiece of modern literature that deserves a place on every bookshelf. Whether you're a seasoned reader or just looking for a captivating story to dive into, this book will not disappoint. Prepare to be transported on an unforgettable journey that will stay with you for a lifetime.	1

Each training observation would be represented by the 5 crafted features shown in the following table.

Features
1. log of Word count
2. number of punctuations (period, comma, apostrophe, quotation, question, exclamation, colon, etc.)
3. number of positive words
4. number of negative words
5. ratio of capitalized words (words starting with capital letter) to total words

Dictionary for positive and negative words is given below:

Positive dictionary
Good, unforgettable, masterpiece, depth, laughed, timeless, captivating, happy, triumph, friendship, modern, lifetime, gripping, enjoy, proud.

Negative dictionary
Not, cannot, disappoint, sad, cried, hopeless, adversity, waste, weird, complexities, anger, seasoned, anticipation, bad, rude.

Your task is to update the weights once, using a stochastic gradient descent algorithm. Assume all initial weights set to 0.2 and learning rate α=0.5. Also compute the Loss function: binary cross entropy loss.

Logistic Regression prediction is given by:

$$\hat{y} = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5)}}$$

Where, $w_0, w_1, \ldots, w_5$ are weights for model
$x_1, \ldots, x_5$ are input features

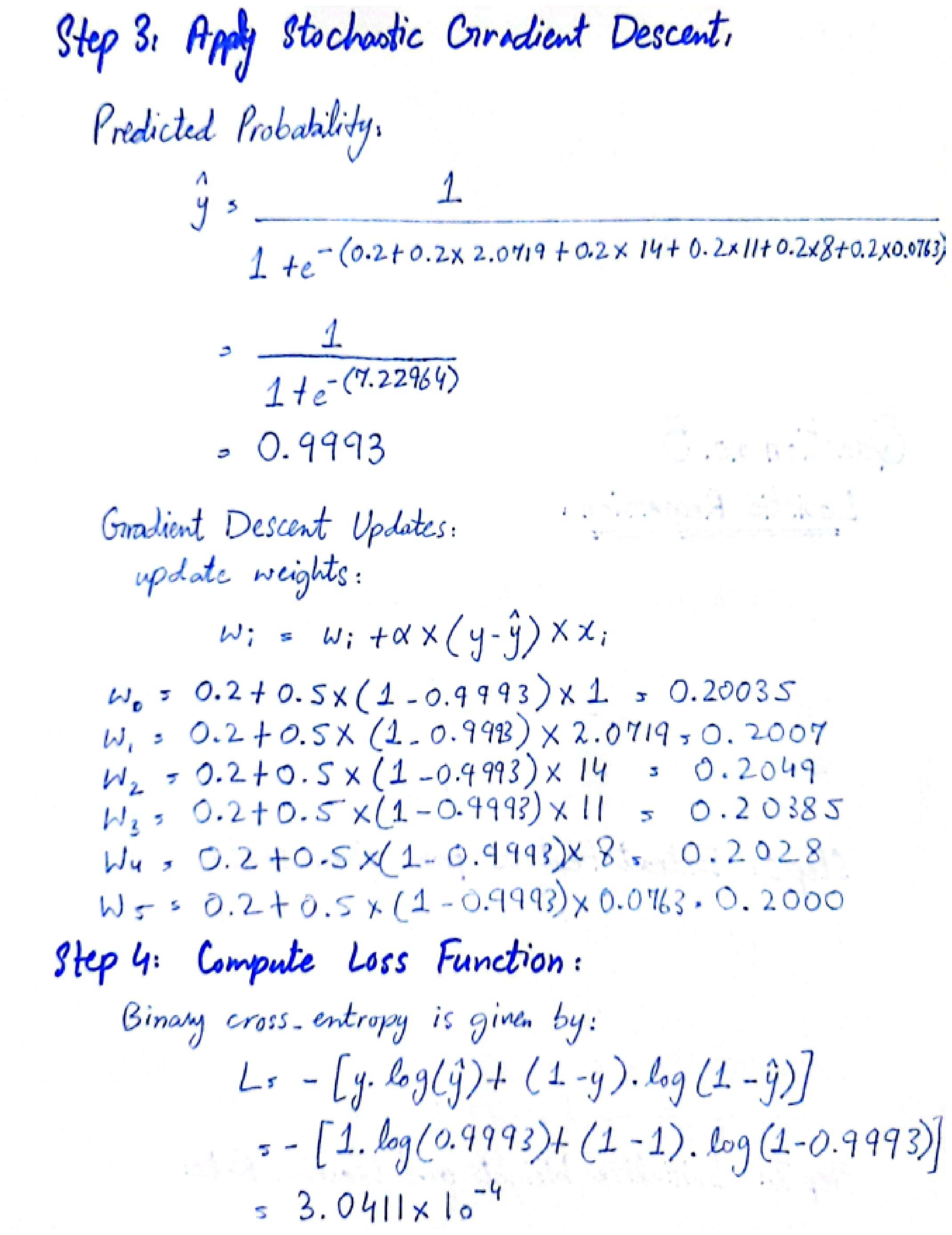
Step 1: Extract Features from given text:

1. log of Word Count: log of 118 $\approx$ 2.0719
2. Number of punctuations: 14
3. Positive Words: 11
4. Negative Words: 8
5. Ratio of Capitalized Words: $9/118 = 0.0763$

Step 2: Initialize Weights and Learning Rate:

$w_0 = w_1 = w_2 = w_3 = w_4 = w_5 = 0.2$

$\alpha = 0.5$



Step 3: Apply Stochastic Gradient Descent:

Predicted Probability:

$$\hat{y} = \frac{1}{1+e^{-(0.2+0.2\times 2.0719+0.2\times 14+0.2\times 11+0.2\times 8+0.2\times 0.0763)}}$$

$$= \frac{1}{1+e^{-(7.22964)}}$$

$$= 0.9993$$

Gradient Descent Updates:

update weights:

$$w_i = w_i + \alpha \times (y-\hat{y}) \times x_i$$

$$w_0 = 0.2+0.5\times(1-0.9993)\times 1 = 0.20035$$

$$w_1 = 0.2+0.5\times(1-0.9993)\times 2.0719 = 0.2007$$

$$w_2 = 0.2+0.5\times(1-0.9993)\times 14 = 0.2049$$

$$w_3 = 0.2+0.5\times(1-0.9993)\times 11 = 0.20385$$

$$w_4 = 0.2+0.5\times(1-0.9993)\times 8 = 0.2028$$

$$w_5 = 0.2+0.5\times(1-0.9993)\times 0.0763 = 0.2000$$

Step 4: Compute Loss Function:

Binary cross-entropy is given by:

$$L = -[y.\log(\hat{y}) + (1-y).\log(1-\hat{y})]$$

$$= -[1.\log(0.9993) + (1-1).\log(1-0.9993)]$$

$$= 3.0411\times 10^{-4}$$

NOTE: Use natural log for loss function.

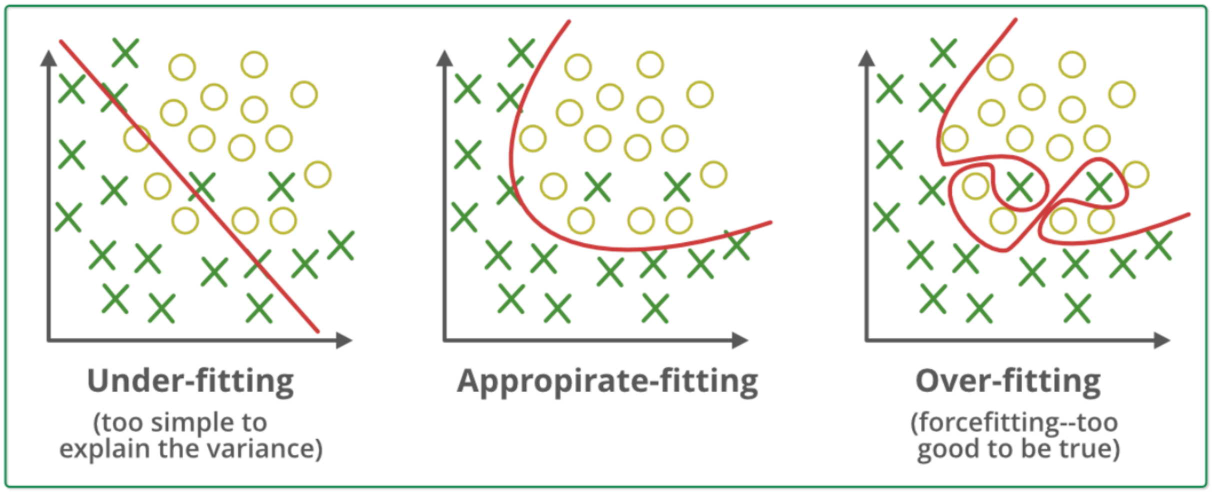
Regularization

Overfitting

- **Definition**: Overfitting occurs when a model performs extremely well on the **training data** but fails to generalize to **new or unseen data (test data)**.
- **Causes**:
  - The model memorizes the training data, including noise, rather than learning the underlying patterns.
  - If a random word only appears in one class, it may receive a **high weight**, which skews predictions on other data without this word.
- **Example**:
  - If a model perfectly predicts a movie review sentiment from words like "drew me in" or "hated", it might overfit to specific combinations of phrases (e.g., 4-gram features) rather than learning general sentiment analysis.
- **Impact**:
  - A model that fits the training data too closely might fail when exposed to novel data points in the test set, leading to **low accuracy** on new data.

Underfitting

- **Definition**: Underfitting occurs when a model is too simple and fails to capture the underlying trends in the data. It results in poor performance on both the training and test datasets.


Under-fitting
(too simple to explain the variance)
Appropirate-fitting
Over-fitting
(forcefitting--too good to be true)

Regularization

- **Purpose**: A technique to reduce overfitting by **penalizing large weights**. The goal is to balance fitting the data well without relying on very large weights.
- **How it works**: A **regularization term** is added to the loss function to penalize large weights and force the model to be simpler. This results in more generalized predictions on unseen data.

Types of Regularization:

1. **L2 Regularization (Ridge Regression)**:
    - **Definition**: Adds the **sum of the squared weights** to the loss function.
    - **Effect**: Reduces the influence of individual weights by penalizing them, preventing the model from giving too much importance to any single feature.
    - **Formula**:

$$J(\theta) = \boxed{\frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]} + \boxed{\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2}$$

    - **Interpretation**: This is the **Euclidean distance** (L2 norm) of the weight vector to the origin, which encourages smaller weight values overall.
2. **L1 Regularization (Lasso Regression)**:
    - **Definition**: Adds the **sum of the absolute values of the weights** to the loss function.
    - **Effect**: Encourages sparsity in the model, meaning it will tend to push some weights to zero, effectively performing **feature selection**.
    - **Formula**:

$$J(\theta) = \boxed{\frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]} + \frac{\lambda}{m}\sum_{j=1}^{n}|\theta_j|$$

    - **Interpretation**: This is the **Manhattan distance** (L1 norm) of the weight vector to the origin. It reduces the number of features in the final model by eliminating some weights.