.............A Must Read............. Logistic Regression: Pseudo-code, Complexity Analysis, and Implementation Details

Tahrima Rahman
The University of Texas at Dallas

Why read this document? To avoid implementing a logistic regression algorithm with a computational complexity of $O(n^2 d)$ per gradient ascent iteration, where n is the number of features and d is the number of training examples. Such complexity is impractical for the spam/ham dataset used in Project 1. Instead, this document presents an algorithm that runs in $O(nd)$ time per iteration.

1 Notation

- **Dataset:** The dataset consists of d training examples and is represented as:

$$\mathbf{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{d}$$

  where:

  - $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)}) \in \mathbb{R}^n$ is the feature vector corresponding to the i-th training example.
  - $x_j^{(i)}$ represents the value of the j-th feature for the i-th training example.
  - $y^{(i)} \in \{0, 1\}$ is the binary label (class) associated with the i-th training example, where 0 and 1 represent two different categories.
  - n is the total number of features.
  - d is the total number of training examples.

- **Weight vector:** The logistic regression model assigns a weight to each feature, represented as:

$$\mathbf{w} = (w_0, w_1, \dots, w_n)$$

  where:

  - w_0 is the bias (intercept) term.

- w_j (for $j = 1, \ldots, n$) is the weight associated with the j-th feature. These weights determine the influence of each feature on the model's decision.

- **Sigmoid function:** Logistic regression uses the sigmoid function to convert a linear combination of features into a probability:

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

where z is a scalar input computed from the dot product between the weight vector $\mathbf{w}$ and any feature vector $\mathbf{x}$. That is $z = \mathbf{w}^T \mathbf{x} = w_0 + \sum_{j=1}^{n} w_j x_j$.

- **Probability model:** Given a feature vector $\mathbf{x}^{(i)} \in \mathbf{D}$, the probability of the corresponding label being 1 is modeled as:

$$P(Y = 1|\mathbf{x}^{(i)}; \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{x}^{(i)}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x}^{(i)})}$$

where:

  - $\mathbf{w}^\top \mathbf{x}^{(i)} = w_0 + \sum_{j=1}^{n} w_j x_j^{(i)}$ is the weighted sum of the features, including the bias term.
  - The sigmoid function ensures that the output is in the range $(0, 1)$, which can be interpreted as a probability.

- **Conditional Log-likelihood Objective Function:** The likelihood function measures how well the model parameters fit the observed data. The log-likelihood function is:

$$\ell_{\mathbf{D}}(\mathbf{w}) = \sum_{i=1}^{d} \left[y^{(i)} \ln P(Y = 1|\mathbf{x}^{(i)}; \mathbf{w}) + (1 - y^{(i)}) \ln(P(Y = 0|\mathbf{x}^{(i)}; \mathbf{w}) \right]$$

where:

  - The first term corresponds to the contribution of samples where $y^{(i)} = 1$.
  - The second term corresponds to the contribution of samples where $y^{(i)} = 0$.
  - Maximizing this function ensures that the model assigns high probabilities to the correct class labels.

- **Gradient update rule for Maximum Conditional Likelihood Estimation (MCLE):** Logistic regression updates the weight vector using gradient ascent, following the rule:

$$w_{j,t+1} = w_{j,t} + \eta \sum_{i=1}^{d} x_j^{(i)} \left(y^{(i)} - P(Y = 1|\mathbf{x}^{(i)}; \mathbf{w}_t) \right)$$

where:

- j indexes the features ($j = 0, 1, \ldots, n$).
- t indexes the iteration number in the optimization process.
- $w_{j,t}$ is the weight corresponding to feature j at iteration t.
- $x_j^{(i)}$ is the value of feature j for the i-th training example.
- $y^{(i)} \in \{0, 1\}$ is the binary label (class) associated with the i-th training example, where 0 and 1 represent two different categories.
- $P(Y = 1|\mathbf{x}^{(i)}; \mathbf{w}_t)$ is the predicted probability that $Y = 1$ for the i-th example using the weight vector at iteration t.
- η is the learning rate, controlling the step size of the update.

- **Gradient update rule for Maximum Conditional a Posteriori (MCAP) with ℓ_2 Regularization:** To prevent overfitting, we introduce an ℓ_2-norm penalty term, modifying the update rule as:

$$w_{j,t+1} = w_{j,t} + \eta \times \left(\sum_{i=1}^{d} x_j^{(i)} \Big(y^{(i)} - P(Y = 1|\mathbf{x}^{(i)}; \mathbf{w}_t) \Big) - \lambda \times w_{j,t} \right)$$

where:

- λ is the regularization coefficient that controls the strength of the penalty on large weight values.
- The term $-\eta\lambda w_{j,t}$ reduces the magnitude of the weight $w_{j,t}$, helping to prevent overfitting by discouraging overly large coefficients.
- Note that, we will not be regularizing the bias term w_0.

2 Pseudo-code for Logistic Regression

Algorithm 1: Logistic Regression Training using Gradient Ascent with ℓ_2 Regularization

Input: Input: Data array X (size $d \times n$), Learning rate η, Max iterations T, Regularization constant λ

Output: Optimized weight array w (size $n+1$)

begin
- Augment X with a column of 1's for the dummy feature X_0 `// size(X):` $d \times (n+1)$
- Initialize weight array w randomly `// size(`w`):` $n+1$
- **for** $t = 1$ *to* T **do**
  - `// Create an array` y_{pred} `to store predictions` $P(Y=1|\mathbf{x}^{(i)}; \mathbf{w}_t)$ `for each sample at each iteration` t
  - Initialize prediction array y_{pred} `// size(`y_{pred}`): d`
  - `// Create an array` z `to store the` $w_0 + \sum_{j=1}^{n} w_j x_j^{(i)}$ `for each sample at each iteration` t
  - Initialize z `// size(z): d`
  - `// Compute predictions for all samples`
  - **for** $i = 1$ *to* d **do**
    - `// Compute the weighted sum for sample` i
    - $z[i] = 0$
    - **for** $j = 0$ *to* n **do**
      - $z[i] = z[i] + w[j] \times X[i][j]$
    - **end**
    - `// Apply the sigmoid function to get probability`
    - $y_{\text{pred}}[i] = \frac{1}{1+e^{-z[i]}}$
  - **end**
  - Initialize gradient vector g `// size(g)=n+1`
  - **for** $j = 0$ *to* n **do**
    - $g[j] = 0$
    - **for** $i = 1$ *to* d **do**
      - `// Gradient of conditional log-likelihood for` w_j
      - $g[j] = g[j] + X[i][j] \times (y[i] - y_{\text{pred}}[i])$
    - **end**
    - `// Apply` ℓ_2 `regularization to all weights but` w_0
    - **if** *$j \neq 0$* **then**
      - $g[j] = g[j] - \lambda \times w[j]$
    - **end**
    - `// Update weights using gradient ascent`
    - $w[j] = w[j] + \eta \times g[j]$
  - **end**
- **end**
- `// Return the optimized weight vector after training`
- Return w

end

3 Computational Complexity Analysis

The computational complexity of logistic regression is determined by the following factors:

- **Computing Predictions:** Each prediction requires $O(n)$ operations per sample. For d samples, this results in $O(dn)$.
- **Gradient Computation:** The gradient update requires summing over d samples, leading to $O(dn)$.
- **Gradient Update:** Updating each weight is $O(n)$, and for T iterations, the overall complexity is:

$$O(Tdn)$$

4 Implementation Details

- **Choice of Learning Rate η:** A large η may cause divergence, while a small η results in slow convergence.
- **Stopping Criteria:** Fixed number of iterations T.