Logistic Regression

Logistic regression is used for binary classification problems. It is **not** used for regression because we are not trying to compute a target value, rather we are trying to compute a classification (0 or 1).

The logistic sigmoid function takes a value between negative and positive infinity and maps it to a value between 0 and 1:

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

In other words, the probability that the output is 1 given input features X is:

$$P(y = 1|X) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

where $z = Xw$ is the linear combination of input features and weights. The output of the sigmoid function can be interpreted as the probability that the input belongs to the positive class (1).

The probability that the output is 0 is:

$$P(y = 0|X) = 1 - \sigma(z)$$

Derivation of the Logistic Regression Cost Function

We want to define the weights that minimize the error, maximizing the likelihood or correct classification. Starting out, we have a case based objective function J that we want to minimize:

$$J = if\ y = 1 : \hat{y}\ else :\ (1 - \hat{y})$$

Where $\hat{y}$ is the predicted probability that $y = 1$.
This is not optimal, so we can use a trick to write this as:

$$J = \hat{y}^y (1 - \hat{y})^{(1-y)}$$

This is called the MLE, or maximum likelihood estimation.
To make this easier to work with, we will take the log of this function (the logarithm is a monotonic function, so maximizing the log of a function is the same as maximizing the function itself):

$$\log J = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$

There are two ways to turn a maximization problem into a minimization problem: we can either negate the function or the take the reciprocal.
To convert this into a minimization problem, we will use the negation:

$$-\log J = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

where $\hat{y} = \sigma(Xw)$.

This is the logistic regression cost function or "log loss" function.
To find the optimal weights, we can take the derivative of this cost function with respect to the weights and use gradient descent to find the weights that minimize the cost. **There is no direct solution**.

Gradient of the Logistic Regression Cost Function

Note: Derivations needed here
The online gradient rule is: $\frac{\partial J}{\partial W} = X^T(\hat{y} - y)$

The batch gradient rule is: $\frac{\partial J}{\partial W} = \frac{1}{N} X^T(\hat{y} - y)$

There are the same update rules as linear regression, just a different way to compute $\hat{y}$

Numeric Instability

Issues can occur if we are dividing by 0 or taking the log of 0. To prevent this, we can add a small value (epsilon) to the input of the log function:

$$-\log J = -y \log(\hat{y} + \epsilon) - (1 - y) \log(1 - \hat{y} + \epsilon)$$

where ϵ is a small constant (e.g., 1×10^{-15}).

We can also use clipping to ensure that $\hat{y}$ is never exactly 0 or 1:

$$\hat{y} = \text{clip}(\hat{y}, \epsilon, 1 - \epsilon)$$

This ensures that $\hat{y}$ is always within the range $[\epsilon, 1-\epsilon]$, preventing issues with $\log(0)$. Clipping is the practice of constraining a value to lie within a specified range that is defined by a minimum and maximum value, keeping it away from 0 and 1.

Dealing with Overfitting

General Approaches

- Get more data for training via cross-validation
- Get more data to improve generalization.
- Don't use all the features

Algorithmic Approaches

- Early stopping (monitoring performance on a validation set and stopping training when performance starts to degrade)
- Stochastic gradient learning where you update weights after each observation (or some number of obersavations) instead of after the whole dataset.
- Add a regularization term to the cost function (L1 or L2 regularization) to penalize large weights (model complexity).