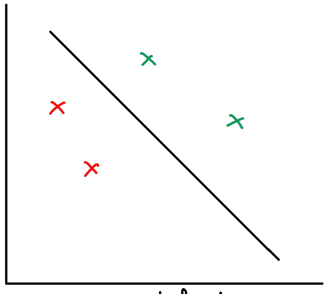
Logistic Regression

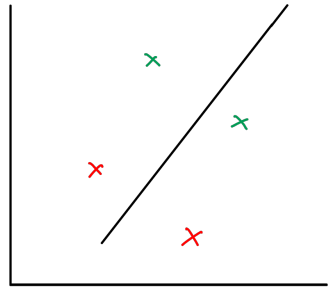
Logistic Regression

Logistic Regression is a statistical method used for binary classification problems. Unlike linear regression which predicts continuous values, logistic regression predicts the probability that an instance belongs to a particular class.

Intuition



Model 1



Model 2

In the above models, it's clear that model 1 is a better one since it's easily able to classify the points.
While in model 2, the model is misclassifying.

Now, if our model has equal number of misclassifications on both sides, it will confuse the model.

How does the point end up there?

In classification, we have to make sure that the +ve points land in +ve region and -ve points in the -ve regions.
If we used the linear regression directly, : $P(y) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$, this would confuse the model.

To avoid this we introduce Sigmoid function.

The sigmoid function maps any real number to a value between 0 and 1, making it perfect for probability estimation.

Sigmoid function will gives us the strength (how far the point is from the distinction line) and we can use that to estimate the exact region the point lies.

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

$z \rightarrow \beta_0 + \beta_1 x_1 + \beta_2 x_2$

↓

This sigmoid gives us the probability of each point.

Now, to calculate the loss function, we calculate the maximum likelihood (since outcome is binary), we multiply each probability value of the given class

The likelihood function is the product of the predicted probabilities for the actual class of each observation.

When we multiply, (ex; $0.6 \times 0.3 \times 0.9 \times 0.4$), the value comes out to be smaller and due to underflow, it would be hard for computer to do the calculations after certain limit, therefore, we introduce logarithmic function called Log Loss.

$$\log(mL) = \log(0.8 \times 0.6 \times 0.9 \times 0.4)$$

and since, the values are inverted, we assign a negative sign in front.

$$\therefore -\log(\hat{y}_1) - \log(\hat{y}_2) - \log(\hat{y}_3) - \log(\hat{y}_4)$$

where $y = \sigma(z) \rightarrow P(\text{green})$ probability of getting green

sigmoid ↓

$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2$

The final formula is setup in a way that if we take value of red(0) or green(1), it will give us either one from red or green.

$$\therefore L = -\frac{1}{n}\sum_{i=1}^{n} y_i(\log \hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)$$

Log Loss Error
or
Binary Cross Entropy

Here $\frac{1}{n}$ is there since we are taking average.

Applying Gradient descent. (This is to find value of $\beta_0, \beta_1, \beta_2$ etc) and we find the best values for $\beta_1 \ldots$)

$$\frac{\partial L}{\partial \beta_1} = -\frac{1}{n}\sum_{i=1}^{n} y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$$

Integrating the formula
(Ignoring n for now) ((considering this only for one point)

$$\frac{\partial L}{\partial \beta_1} = \frac{\partial}{\partial \beta_1}\left[-y\log\hat{y} - (1-y)\log(1-\hat{y}) \right]$$

$-y\,\sigma(z)[1-\sigma(z)]x_1$

$-y\,\hat{y}(1-\hat{y})x_1$

$-y(1-\hat{y})x_1$

$\frac{(1-y)}{(1-\hat{y})}\,\hat{y}(1-\hat{y})x_1$

$(1-y)\hat{y}x_1$

$$= -y(1-\hat{y})x_1 + (1-y)\hat{y}x_1$$

$$\therefore \boxed{\frac{\partial L}{\partial \beta_1} = (\hat{y}_i - y_i)x_{1i}}$$

Quick Summary (Pipeline)

Sigmoid : Maps any real number to probability $(0,1)$

↓

Training Maximum Likelihood Estimation : MLE finds paramaters that makes our observed data most probable.

↓

Optimization log likelihood or Binary cross entropy : Takes log of likelihood for easier computation.
Maximizing log likelihood = Minimizing Binary cross entropy
Loss = $-\frac{1}{n}\sum_{i=1}^{n} y \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$

↓

Solving Gradient Descent : Uses iterative method to find the optimal $\beta_0, \beta_1, \beta_2$ etc.
Gradient descent minimizes the cross entropy loss func.

The reason we use gradient descent is that we will need to find the best values for $\beta_0, \beta_1, \beta_2$ to minimize the Loss function.

Multiclass Classification using Logistic Reg.

Two approach
- OVR (One vs Rest) LR
- Multinomial (Softmax) LR

OVR Approach

One vs- Rest or One vs All (OVA) is a strategy for extending binary classification algorithms to handle multi-class problems. It's particularly useful for logistic regression, which is naturally a binary classifier.

Intuition

i. Decompose the multi-class problem into binary classification
- We generally use OneHotEncoder to convert each class into binary format.

ex.

iq	gpa	job
-	-	yes
-	-	no
-	-	opt-out

job-yes	job-No	job-optout
1	0	0
0	1	0
0	0	1

↙ cols

If you notice, the new cols are now in binary format in itself. The date is separated based on each class.

ii. for k classes, create k binary classifiers
- Now we train one model for each class which gives us the probability of each class vs rest classes.

iii. Each class distinguishes one class from all the other class
- Calculate the probability and finalize.

for prediction, it calculates the average probability

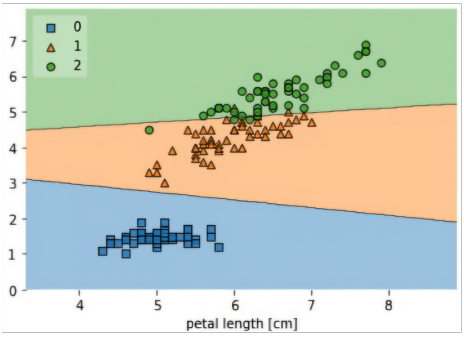
Ex. Yes = 0.3
No = 0.45
opt out = 0.25

Yes → $\frac{0.3}{0.3+0.45+0.25}$

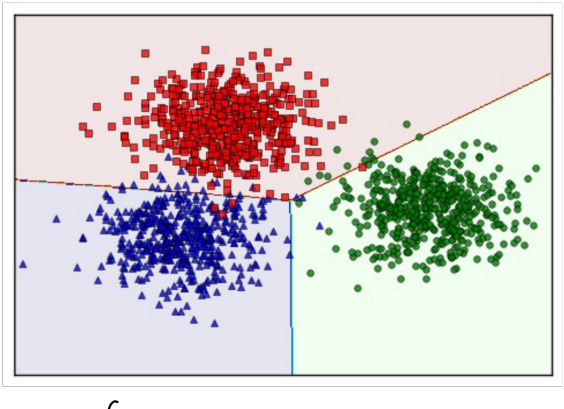
opt out → $\frac{0.25}{0.3+0.45+0.25}$

No → $\frac{0.45}{0.3+0.45+0.25}$

The maximum's respective class is assigned to the data point.



Multinomial Logistic Regression (Softmax)



Softmax function

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

ex:

$\sigma(z_1) + \sigma(z_2) + \sigma(z_3)$

↓

$$\sigma(z_1) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Loss function

$$L = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_i^k \log \hat{y}_i^k$$

Classification Cross entropy

(This is standard formula)

The loss function formula for OVR is a special format, where $k = 2$, which results in

$$L = y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i)$$

$$L = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_i^K \log \hat{y}_i^K$$

This formula is designed in a way, that the value will always be for one class(i) and rest will be 0.

ex: yes → $\log(\hat{y}^{yes}) + 0 + 0$

No → $0 + \log(\hat{y}^{No}) + 0$

opt → $0 + 0 + \log(\hat{y}^{opt})$

$\therefore L = \log(\hat{y}^{yes}) + \log(\hat{y}^{No}) + \log(\hat{y}^{opt})$

$\hat{y}^{yes}$: Output of logistic Reg

$\hat{y}^{yes}$: $\sigma(z_i) \rightarrow \frac{e^{z_{yes}}}{e^{z_{yes}} + e^{z_{No}} + e^{z_{opt}}}$

z_{yes} : $\beta_0 + \beta_1 x_1 + \beta_2 x_2$

x_1, x_2 : value of input col in the data point.

β

$\beta_0 = n \frac{\partial L}{\partial \beta_0}$ → β value obtained via Gradient descent

Softmax creates k lines for k classes. Each line has different values for β_0, β_1, β_2 etc which is obtained from gradient descent.

When to Use Which one?

Use One-vs-Rest (OVR) When:

- Classes are Non-Mutually Exclusive:
  OVR is suitable if an instance can belong to more than one class, as each classifier provides an independent probability for each class.
- Dealing with Imbalanced Data:
  OVR may perform better when class distribution is highly imbalanced since each class gets a dedicated model.

Use Multinomial Logistic Regression (Softmax Regression) When:

- Computational Efficiency is Required:
  Softmax Regression is generally more efficient for large datasets and a high number of classes.
- Classes are Mutually Exclusive:
  Softmax Regression is ideal when each instance can only belong to one class. The Softmax function provides a set of probabilities that sum to 1, which fits well with mutually exclusive classes.
- Interpretability is Important:
  The probabilities output by Softmax Regression are more interpretable than those from OVR, as they always sum to 1. This makes model predictions easier to explain.

Assumptions of Logistic Regression

Logistic regression, like other statistical methods, relies on certain assumptions. Here are the main assumptions of logistic regression:

1. Binary Logistic Regression requires the dependent variable to be binary: That means the outcome variable must have two possible outcomes, such as "yes" vs "no", "success" vs "failure", "spam" vs "not spam", etc.

2. Independence of observations: The observations should be independent of each other. In other words, the outcome of one instance should not affect the outcome of another.

3. Linearity of independent variables and log odds: Although logistic regression does not require the dependent and independent variables to be related linearly, it requires that the independent variables are linearly related to the log odds.

4. Absence of multicollinearity: The independent variables should not be too highly correlated with each other, a condition known as multicollinearity. While not a strict assumption, multicollinearity can be a problem because it can make the model unstable and difficult to interpret.

5. Large sample size: Logistic regression requires a large sample size. A general guideline is that you need at least 10 cases with the least frequent outcome for each independent variable. For example, if you have 5 independent variables and the expected probability of your least frequent outcome is 0.10, then you would need a minimum sample size of 500 (10*5 / 0.10).

Note that violating these assumptions doesn't mean you can't or shouldn't use logistic regression, but it may impact the validity of the results and you should proceed with caution.

Regularization on Logistic Regression

Regularization is a technique used in machine learning models to prevent overfitting, which occurs when a model learns the noise along with the underlying pattern in the training data. Overfitting leads to poor generalization performance when the model is exposed to unseen data.

In the context of linear models like linear regression and logistic regression, regularization works by adding a penalty term to the loss function that the model tries to minimize. This penalty term discourages the model from assigning too much importance to any single feature, which helps to prevent overfitting.

The most common types of regularization in linear models are L1 and L2 regularization:

1. L1 regularization (Lasso Regression): This technique adds a penalty term equal to the absolute value of the magnitude of the coefficients. Mathematically, it's represented as the sum of the absolute values of the weights (||w||_1). This can lead to sparse models, where some feature weights can become exactly zero. This property makes L1 regularization useful for feature selection.
2. L2 regularization (Ridge Regression): This technique adds a penalty term equal to the square of the magnitude of the coefficients. Mathematically, it's represented as the sum of the squared values of the weights (||w||_2^2). L2 regularization tends to spread the weight values more evenly across features, leading to smaller, but non-zero, weights.

There's also Elastic Net regularization, which is a combination of L1 and L2 regularization. The contribution of each type can be controlled with a separate hyperparameter.

In all these techniques, the amount of regularization to apply is controlled by a hyperparameter, often denoted as λ (lambda). Higher values of λ mean more regularization, leading to simpler models that might underfit the data. Lower values of λ mean less regularization, leading to more complex models that might overfit the data. The optimal value of λ is typically found through cross-validation.