Logistic Regression

Başlamadan önce öğrenmemiz gereken konseptlerden bahsedelim;

odds and Log odds :

The **odds** are the ratio of something happening (ex: my team winning) to something not happening (ex: my team not winning).

ÖR: Örneğin; 8 maç oynadım → 5 Galibiyet, 3 Mağlubiyet

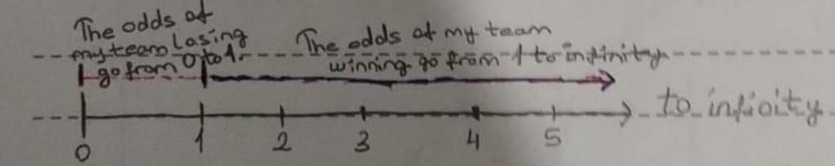
1. yöntem

odds of winning $= \frac{5}{3} = 1.7$

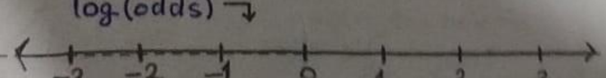
2. yöntem

$$\text{odds of winning} = \frac{\text{The probability of winning}}{(1-\text{the probability of winning})} = \frac{5/8}{3/8} = \frac{5}{3} = 1.7$$

$$\text{odds of winning} = \frac{p}{1-p} = \frac{5/8}{3/8} = 1.7$$



Taking the log() of the odds solves that asimetri problem by making everything symmetrical



$$\log(\text{odds}) = \log\left(\frac{5}{3}\right) = \underbrace{\log\left(\frac{p}{1-p}\right)}_{\text{logit function}} = \log(1.7) = 0.53$$

odds ratios and log(odds ratios):

Even though the "odds" is a ratio; it is different from an "odds ratio".

When people say "odds ratio", they are talking about a "ratio of odds".

$$= \frac{3}{2} \quad , \quad = \frac{2}{4}$$

$$\text{odds Ratio} = \frac{3/2}{2/4} = 3$$

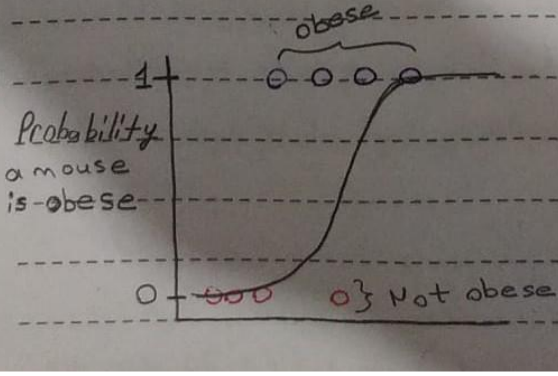
log(odds Ratio) ⟹ yine aynı şekilde kullanılır

		Has Cancer: Yes	Has Cancer: No
Mutated Gene	Yes	23	117
	No	6	210

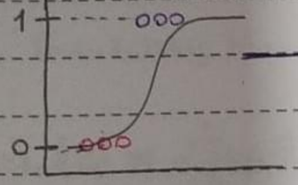
$$\Rightarrow \text{Odds Ratio} = \frac{\frac{23}{117}}{\frac{6}{210}} = 6.88$$

The odds are 6.88 times greater that someone with the mutated gene will also have cancer.

Başlayalım;

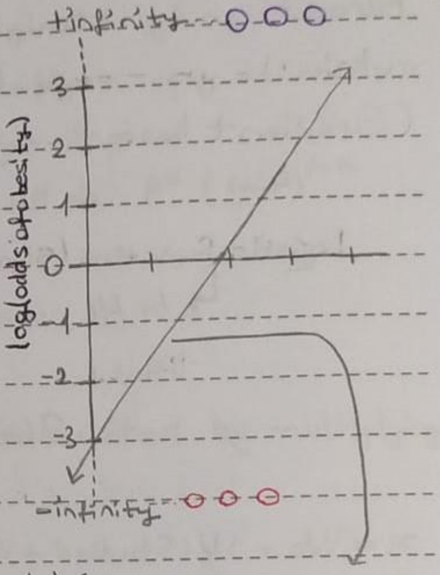


$$\log(\text{odds of obesity}) = \underbrace{\log\left(\frac{p}{1-p}\right)}_{\text{Logit function}}$$



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

The new y-axis transforms the S line into straight line.



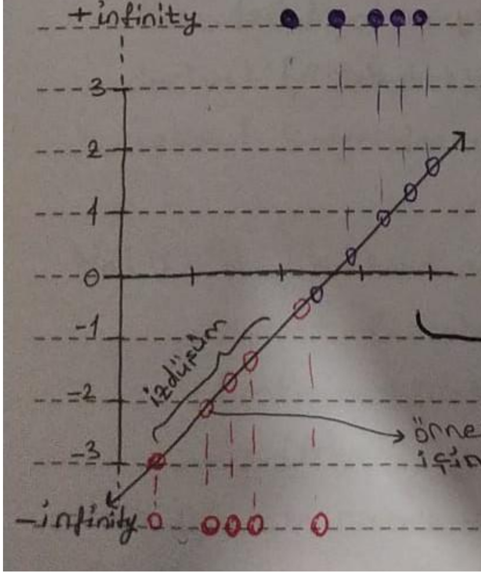
$\boxed{\text{logit}(p_i) = y = -3.48 + 1.83 \text{weight}}$ (log-odds)

it means that for every one unit of weight gained, the log(odds of obesity) increases by 1.83.

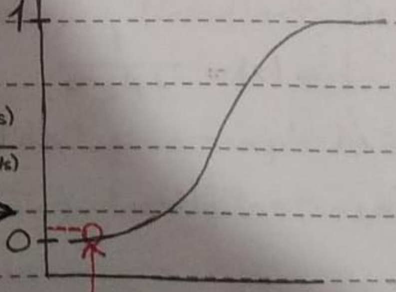
Maximum Likelihood Logistic Regression;

S eğrisinin en iyi nasıl optimize edildiğine bakacağız.

$$p = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$ (logistic or Sigmoid func.)



$$p = \frac{e^{\log(\text{odds})}}{1 + e^{\log(\text{odds})}}$$



örneğin bu nokta için;

$$p = \frac{e^{-2.1}}{1 + e^{-2.1}} = 0.1$$ (Obezite olasılığı 0.1)

../..../....

* Buradaki en iyi straight line Maximum Likelihood'u maksimum yapan veya log loss'u minimum yapan line'dır. (Gradient Descent veya Ascent ile bu line bulunur)

Logistic function (Sigmoid func)

↳ In ML, we use it to map predictions to probabilities.

$$S(z) = \frac{1}{1+e^{-z}}$$; $S(z) \Rightarrow$ output between 0 and 1. (probability)

z = input to the function (algorithm prediction e.g $= mx+b$)

$z = W_0 + W_1 \text{Studied} + W_2 \text{Slept}$

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

↳ if it returns 0.4; then only %40 chance of passing.

(Denklemden gelen output'u Sigmoid Function yardımı ile olasılık değerine dönüştürürüz.

Cost function; (Cross Entropy or log loss)

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))]$$

$$h_\theta(x) = \frac{1}{1+e^{-z}}$$

Parametre Seçimi;

We use Maximum likelihood estimation.

Ondan önce hatırlamamız gereken konseptler var;

Probability Mass function of Bernoulli $\Rightarrow$ $p^x (1-p)^{1-x}$

$$L(\theta) = \prod_{i=1}^{n} p_i^{x_i} (1-p_i)^{1-x_i}$$

The likelihood of all the data is calculated by multiplying each individual binomial probabilities.

log'unu aldığımızda, we get log likelihood for logistic Regression:

$$\sum_{i=1}^{n} y_i \cdot \log(p(y_i)) + (1-y_i) \log(1-p(y_i))$$

Amacımız, öyle parametreler seçmek ki (θ_j) bu log likelihood fonksiyonunu maksimize etsin veya diğer bir deyişle Cross entropy (log loss) yani cost function'u minimize etsin.

Bunun için aynı Liner Regresyonda olduğu gibi partial derivative'ler alınarak en iyi straight line'a yani en optimal S eğrisine ulaşmış olacağız. (Gradient Descent).

Not: The likelihood of parameter θ given sample X is the product of probability density family instances, for Continious Distribution.

$$\mathcal{L}(\theta) = \prod_{t=1}^{N} p(x^t | \theta)$$

For Discrete Distribution, $\mathcal{L}(\theta) \Rightarrow$ product of probability mass family instances.

General formulas for Logistic Regresyon

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$$ } logistic function (Sigmoid)

After manipulation $\overbrace{\frac{p(X)}{1-p(X)}}^{\text{odds}} = e^{\beta_0 + \beta_1 X}$

Take logarithm of both side; $\underbrace{\log\left(\frac{p(X)}{1-p(X)}\right)}_{\text{log-odds(logit)}} = \beta_0 + \beta_1 X$

Increasing X by one unit changes the log odds by β_1.

Making Prediction;

$$\hat{p}(X) = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 X}}{1+e^{\hat{\beta}_0+\hat{\beta}_1 X}}$$

Multiple logistic Regression;

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p$$

$$p(X) = \frac{e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1 + \dots + \beta_p X_p}}$$

Multiclass Logistic Regression (3 farklı class var ise veya daha fazla)

One vs All (one vs Rest)

Diyelim ki 3 sınıfımız var; bu durumda 3 farklı binary classification yaparız.

→ Her seferinde sınıflardan birine 1, diğerlerine 0 verilerek 3 farklı binary problem yaratılır.

* Sonuç olarak, bir gözlemin hangi class'a düşeceğini belirlemek için Bu 3 classifier'ı da run ederiz ve Probability'i max çıkan sınıfa atarız. (Pick the class i that maximize $h_\theta^{(i)}(x)$.)

Logistic Regression Parameters:

1) penalty : {"l1", "l2", "elasticnet", "none"} ; Default = "l2"

Log-loss function:

$$L_{log} = -\ln(L) = -\sum_{i=1}^{n} \left[-\ln\left(1+e^{(\beta_0+\beta_1 x_i)}\right) + y_i(\beta_0+\beta_1 x_i) \right]$$

Eğer modelimiz training sette iyi, test sette kötü sonuç veriyorsa "Overfit" ediyor demektir.

Yani; "We need regularization to introduce Bias to the model and to decrease the variance".

Bu durumda penalty parametresi yardımıyla penalization yapmış oluruz.

3 yöntem kullanılır: Bu yöntemler Loss Function'a penalty terimler eklenerek yapılır.

1) L_2 Norm $\Rightarrow L_{log} + \lambda \sum_{j=1}^{p} \beta_j^2$

2) L_1 Norm $\Rightarrow L_{log} + \lambda \sum_{j=1}^{p} |\beta_j|$

3) Elasticnet $\Rightarrow L_{log} + \lambda \sum_{j=1}^{p} (\alpha \beta_j^2 + (1-\alpha)|\beta_j|)$

↳ Use both L1 and L2 penalties.

2) C ; Default = 1.0

Inverse of regularization strength.

We use "C" parameter as our regularization parameter.

$C = 1/\lambda$.

* Örneğin; if λ is very low or 0, the model will have enough power to increase its complexity (overfit) by assigning big values to weights for each parameter. On the other hand, if we increase the value of λ, the model will tend to underfit as the model will become too simple.

Bu yapıldığında $\quad$ Bunlar olur.

Yani; $\underbrace{\lambda \downarrow}$ $\underbrace{\beta_j \uparrow \text{ Overfit} \uparrow}$

$\lambda \uparrow$ $\beta_j \downarrow$ Underfit $\uparrow$

$C \downarrow$ $\beta_j \downarrow$ Underfit $\uparrow$ } we increase regularization strength by C↓.

$C \uparrow$ $\beta_j \uparrow$ overfit $\uparrow$ } we low the power of regularization.

Sonuç olarak λ varyans-Bias trade-off'unu kontrol eder

ve $C = 1/\lambda$.

3) tol, Default = 1e-4

Tolerance for stopping criteria.

* During the training, algorithm tries to minimize Loss.

It always checks its convergence on computing the difference between loss at present iteration to its previous iteration.

This residual value is called Tolerance.

*(Eğer algoritma verilen tolerans değeri ile karşılaşırsa o iterasyonda train'i durdurur.)

Tolerance = (loss at n^{th} iteration) - (loss at $(n-1)^{th}$ iteration)

örnek olarak

Yüksek Tolerans değeri verirsek, eğitim erken bitecek ve kötü bir sınıflandırma yapmış olacağız.

4) fit-intercept, Default = True

fit-intercept = False olduğu zaman Doğrumuz orijin'den geçer ve intercept = 0 olur.

5) intercept-scaling, Default = 1

Intercept scaling değeri arttıkça intercept'in değeri artar ancak bağımsız değişken weight'leri azalır ve etkileri düşer.

6) class_weight, default = None

Imbalance data durumunda over sampling / under sampling teknikleri kullanılır. Ayrıca "class_weight" parametresi ile de bu problem çözülebilir.

Not: imbalance data durumunda F1 skor ile başarı ölç.

→ Buradaki problem şuradan çıkar: Minority class'ı iyi öğrenemeyiz ve minority class'ları sınıflandırmada çok başarısız oluruz.

"class_weight" parametresi ile bu problemi çözeriz.

Eğitim sırasında, algoritmanın cost fonksiyonunda minority sınıfa daha fazla ağırlık veriyoruz, böylece minority sınıfa daha yüksek bir ceza verebilsin ve algoritma minority sınıf için hataları azaltmaya odaklanabilsin.

Manuel bir şekilde weight'ler verilebileceği gibi;

Class_weights = "balanced" yapıldığı durumda, model otomatik olarak

$$w_j = \text{Total gözlem sayısı} \,/\, (\text{Sınıf sayısı} * \text{ilgili sınıftaki gözlem sayısı})$$

formülü ile weight'leri verir.

Formülizasyon:

After Adding Weights

$$\text{Log loss} = \frac{1}{N}\sum_{i=1}^{N}\left[-(y_i * \log(\hat{y}_i) + (1-y_i) * \log(1-\hat{y}_i)\right]$$

$$\text{Log loss} = \frac{1}{N}\sum_{i=1}^{N}\left[-(w_0(y_i * \log(\hat{y}_i)) + w_1((1-y_i) * \log(1-\hat{y}_i))\right]$$

7) solver , default = 'liblinear'

Sklearn solve the objective function in different ways. It uses different algorithms for the same optimization.

Solver options : a) Newton-cg ⇒ it uses exact Hessian matrix

b) lbfgs ⇒ It stores only last few updates, so it saves memory.

c) liblinear ⇒ Uses a coordinate descent algorithm.

d) Sag ⇒ Stochastic Average Gradient Descent.

e) Saga ⇒ Extension of sag that allows L1 regularization.

1) When you have ; Large Dataset use "sag" or "saga" solver.
2) "liblinear" and "saga" handle L1 penalty.
3) "Saga" also supports elasticnet penalty.
4) For small dataset "liblinear" is good choice.

8) max-iter , Default = 100

Solver'lar için maksimum iterasyon sayısı. Yani Log Loss fonksiyonu için maksimum kaç iterasyon yapılma sayısıdır.

9) l1-ratio , default = None

The elasticnet mixing parameter.

Örneğin; setting l1-ratio = 0 is equivalent to using "penalty" = 'l2'.

→ For $0 < l1\text{-ratio} < 1$, the penalty is combination of L1 and L2.