1 Module 13: Logistic Regression

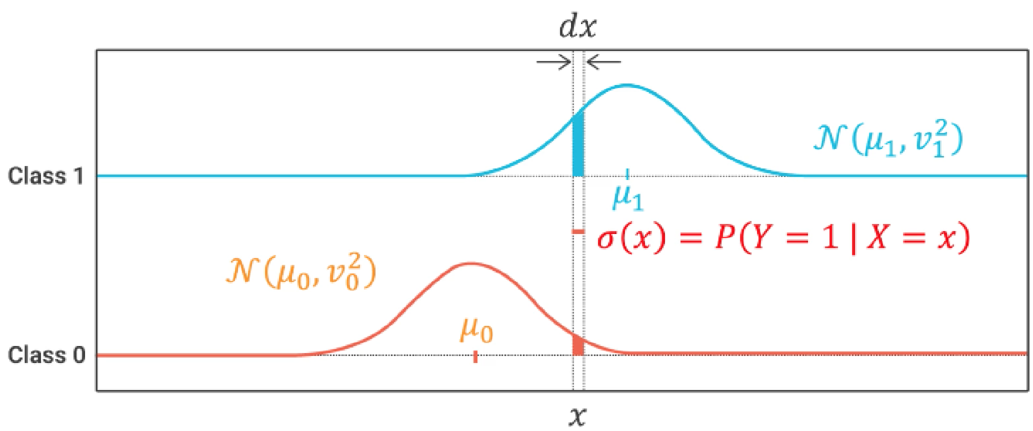
Contents

1.1 Two-class, one-feature sigmoid

For some binned data x with classifications y_1, y_2, the probability of classification in each individual bin is the ratio of the number of examples in each

$$P(Y = y_1 | bin = x_i) = \frac{N(y_1)_i}{N(Y)_i}$$

We can "assume" that the variances of the distributions of y_1, y_2 are the same – so model them with normal curves of same shape & different means.



$$p(X|Y = 0) \sim \mathcal{N}(\mu_0, v_0^2)$$

$$p(X|Y = 1) \sim \mathcal{N}(\mu_1, v_1^2)$$

Odds ratio: the basic ratio which influences the probability $P(Y|X = x) = \sigma(x) = \frac{A}{A+B} = \frac{1}{1+(A/B)}$ where A/B is the odds ratio.

For a Gaussian with common variance ν^2,

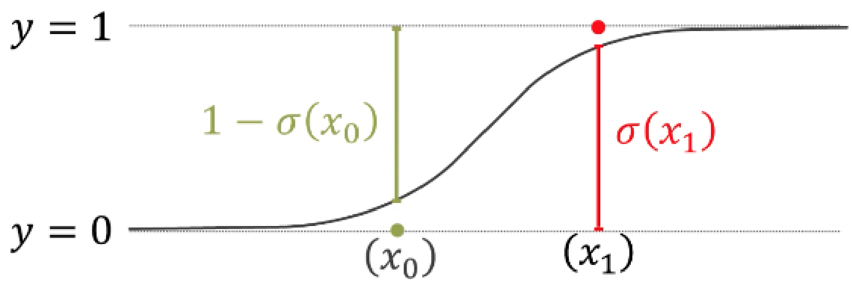
$$\begin{aligned} \frac{P(X=x)|Y=0}{P(X=x|Y=1)} &= \frac{(\sqrt{2\pi\nu})^{-1} exp\left(-\frac{(x-\mu_0)^2}{2\nu^2}\right)}{(\sqrt{2\pi\nu})^{-1} exp\left(-\frac{(x-\mu_1)^2}{2\nu^2}\right)} \\ &= ... = exp\left[-\frac{\mu_1-\mu_0}{\nu^2}x - \frac{\mu_0^2-\mu_1^2}{2\nu^2}\right] \\ &\propto e^{-z} \end{aligned} \tag{1}$$

$$\text{for } z = \beta_0 + \beta_1 \text{ with } \beta_0 = \frac{\mu_0^2-\mu_1^2}{2\nu^2},\ \beta_1 = \frac{\mu_1-\mu_0}{\nu^2}$$

Therefore the Sigmoid Function (or Logistic Function)

$$\sigma(x) = \frac{1}{1+\text{odds ratio}} = \frac{1}{1+e^{-z}}$$

We can approximate the solution with average quantities for $\hat{\mu_0}, \hat{\mu_1}, \hat{\nu}, \hat{\beta_0}, \hat{\beta_1}$
Quantify likelihood of sigmoid given data with



Likelihood:

$$\mathcal{L}(\beta_0, \beta_1) = \prod_{\text{Green}} (1 - \sigma(x_i)) \prod_{\text{Red}} \sigma(x_i)$$

$$= \prod_{i=1}^{N} (1 - \sigma(x_i))^{(1-y_i)} \sigma(x_i)^{y_i}$$

$$log(L(\beta_0, \beta_1)) = log\left(\sum_{i=1}^{N} (1 - \sigma(x_i))^{(1-y_i)} \sigma(x_i)^{y_i}\right) = \\ ... = \sum_{i=1}^{N} ((1 - y_i) log(1 - \sigma(x_i)) + y_i log(\sigma(x_i))) \tag{2}$$

= **Cross Entropy**: minimize for optimum value

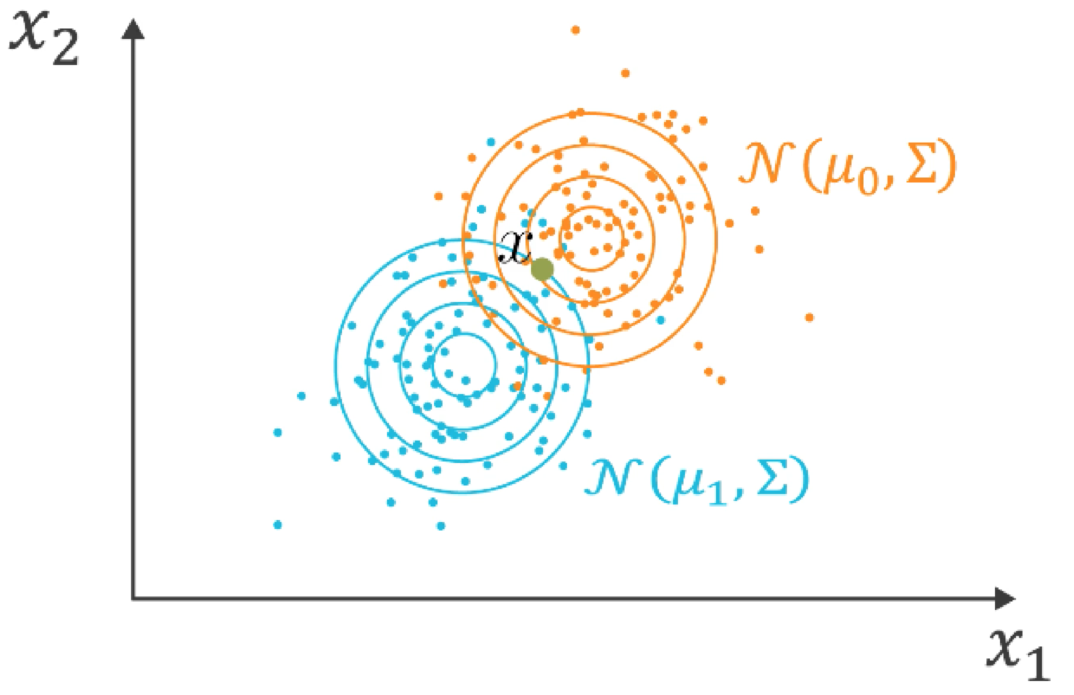
```python
# Select two classes e.g. iris dataset
iris = iris[ (iris.species == 1) | (iris.species == 2) ]

from sklearn.linear_model import LogisticRegression

LogisticRegression().fit(X, y)

# Inspect fit results
beta0 = lr.intercept_[0]
beta1 = lr.coef_[0,0]
threshold = -beta0 / beta1
```

1.2 Multi-feature Sigmoid

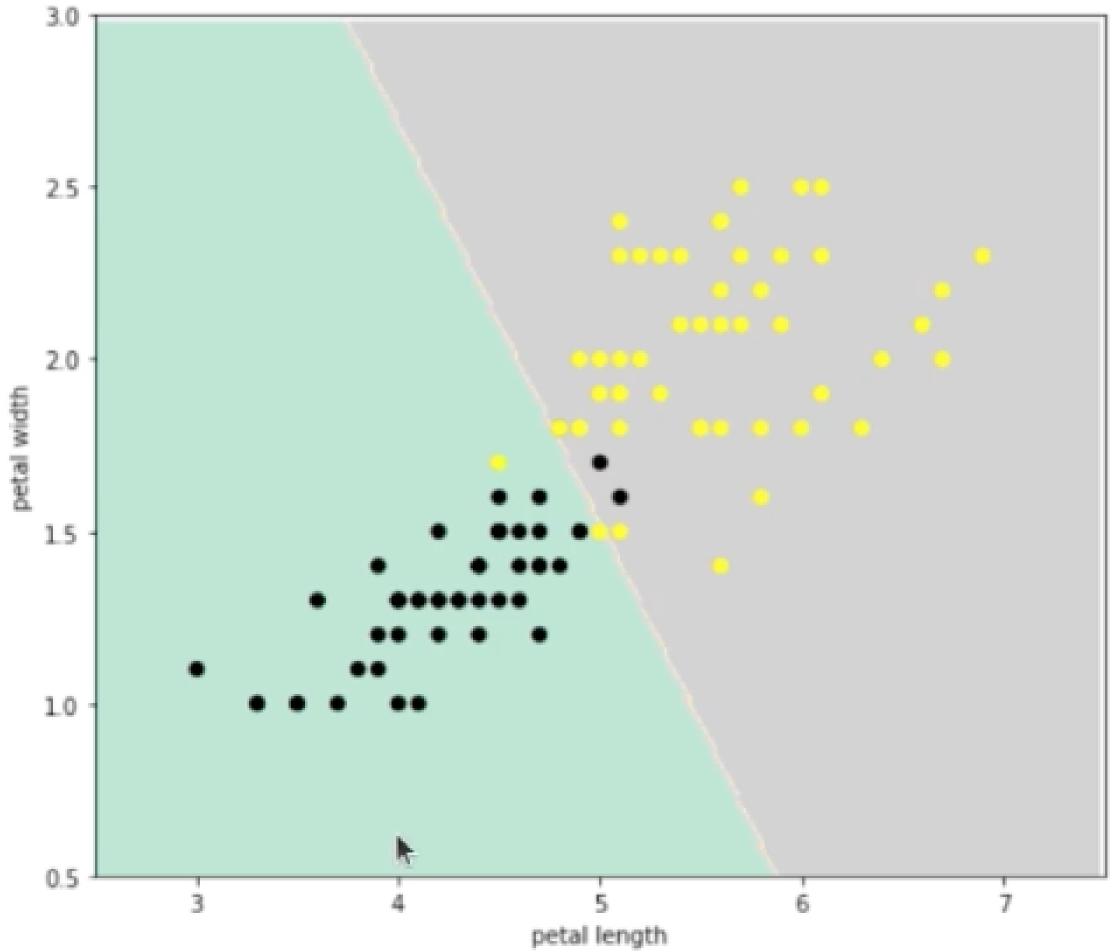


Just add terms to the log odds:

$$z = -\beta_0 - \beta_1 x_1 - \beta_2 x_2 \rightarrow \beta_0 + \sum_{i=1}^{M} \beta_i x_i$$

Minimization be done with L1 or L2 regularization

$$\begin{aligned} \text{minimize} &\rightarrow CE(\text{data}, \beta) + \lambda \sum_{j=1}^{M} |\beta_j| \\ \text{minimize} &\rightarrow CE(\text{data}, \beta) + \lambda \sum_{j=1}^{M} \beta_j^2 \end{aligned} \tag{3}$$



1.3 Multi-class Sigmoid

Notably, multiclass Logistic Regression does not require linearity/normality/homoscedasticity of classes, and teh data can be continous or dichotomous (binary).

Three general approaches for binary classification of multiclass problems:

- One-vs.-one method
  Run binary classification for each possible pair of classes: $K\frac{K-1}{2}$ (unable to classify all data pts.)
- One-vs.-rest method
  Grows linearly with # classes K, but is imbalanced & relies on a continuous probability

- Multinomial regression method

Multinomial regression method (most general form of logistic regression)
Algorithm:

- Specify a reference class (K) as a class to which all others are compared.
- For each other class, build a logistical model

$$\text{1 vs. K: } log\frac{P(Y=1)}{P(Y=K)} = -\beta_{10} - \sum_{j=1}^{M} \beta_{1j}x_j = -\beta_1 x$$

- Iterate through classes 1, 2, ..., K-1 vs. K.
- The probabilities all add up to 1 so, for the other j classes:

$$P(Y=K) = 1 - \sum_{j=1}^{K-1} P(Y=j)$$

- As such, for each other class j and the original K:

$$P(Y=K) = \left(1 + \sum_{j=1}^{K-1} e^{-\beta_j x}\right)^{-1}$$

$$P(Y=j) = \frac{e^{-\beta_j x}}{\left(1 + \sum_{j=1}^{K-1} e^{-\beta_j x}\right)} \text{ for } j = 1...K$$

```
LogisticRegression(multi_class = 'ovr').fit(X, y) # one-vs.-rest (also,
↪  'ovo')

LogisticRegression(multi_class = 'multinomial').fit(X, y) # multinomial

LogisticRegression(penalty = 'l1', C = int \ # Penalty strength
solver = 'liblinear'
).fit(X, y) # Can incl. regularization

# COOL NEW TRICKS

plt.gca().invert_xaxis() # inverts x-axis
```

```
DataFrame.map({'value_in_df': 1, 'another_value_in_df': 2})

list.ravel() # flattens a multidimensional array i.e.
# [[1,2,3],
# [4,5,6]] --> [1,2,3,4,5,6]

np.where(condition, 1, 0) # Returns 1 where condition is True; 0 if False

# Display a confusion matrix/roc from a fitted model (provide ax)
from sklearn.metrics import ConfusionMatrixDisplay, RocCurveDisplay
ConfusionMatrixDisplay.from_estimator(model, X_test, y_test, ax = ax)
```