

In a logistic regression model, we use a weighted sum of these two predictors to calculate a joint score S :

$$S = \beta_0 + \beta_1 X_1 + \beta_2 X_2. \quad (1)$$

The logistic regression model gives us appropriate values for the parameters $\beta_0, \beta_1, \beta_2$ using two sets of example genes:

- OP: Adjacent genes, on the same strand of DNA, known to belong to the same operon;
- NOP: Adjacent genes, on the same strand of DNA, known to belong to different operons.

```
>>> xs = [[-53, -200.78],
           [117, -267.14],
           [57, -163.47],
           [16, -190.30],
           [11, -220.94],
           [85, -193.94],
           [16, -182.71],
           [15, -180.41],
           [-26, -181.73],
           [58, -259.87],
           [126, -414.53],
           [191, -249.57],
           [113, -265.28],
           [145, -312.99],
           [154, -213.83],
           [147, -380.85],
           [93, -291.13]]
```

```
>>> ys = [1,
           1,
           1,
           1,
           1,
           1,
           1,
           1,
           1,
           1,
           1,
           0,
           0,
           0,
           0,
           0,
           0,
           0,
           0]
```

```
>>> model = LogisticRegression().train(xs, ys)
```

Here, xs and ys are the training data: xs contains the predictor variables for each gene pair, and ys specifies if the gene pair belongs to the same operon (1, class OP) or different operons (0, class NOP). The resulting logistic regression model is stored in `model`, which contains the weights β_0 , β_1 , and β_2 :

```
>>> model.beta
[8.9830290157144681, -0.035968960444850887, 0.02181395662983519]
```

Note that β_1 is negative, as gene pairs with a shorter intergene distance have a higher probability of belonging to the same operon (class OP). On the other hand, β_2 is positive, as gene pairs belonging to the same operon typically have a higher similarity score of their gene expression profiles. The parameter β_0 is positive due to the higher prevalence of operon gene pairs than non-operon gene pairs in the tT5Td(1)TJF89.r.5eT5Tps.

```
print "Iteration: ", iteration, "Log-likelihood function: ", loglikelihood
>>>
>>> model = LogisticRegression.train(xs, ys, update_fn=show_progress)
Iteration: 0 Log-likelihood function: -11.7835020695
Iteration: 1 Log-likelihood function: -7.15886767672
Iteration: 2 Log-likelihood function: -5.76877209868
Iteration: 3 Log-likelihood function: -5.11362294338
Iteration: 4 Log-likelihood function: -4.74870642433
Iteration: 5 Log-likelihood function: -4.50026077146
Iteration: 6 Log-likelihood function: -4.31127773737
Iteration: 7 Log-likelihood function: -4.16015043396
Iteration: 8 Log-likelihood function: -4.03561719785
Iteration: 9 Log-likelihood function: -3.93073282192
Iteration: 10 Log-likelihood function: -3.84087660929
Iteration: 11 Log-likelihood function: -3.76282560605
Iteration: 12 Log-likelihood function: -3.69425027154
Iteration: 13 Log-likelihood function: -3.6334178602
Iteration: 14 Log-likelihood function: -3.57900855837
Iteration: 15 Log-likelihood function: -3.52999671386
Iteration: 16 Log-likelihood function: -3.48557145163
Iteration: 17 Log-likelihood function: -3.44508206139
Iteration: 18 Log-likelihood function: -3.40799948447
Iteration: 19 Log-likelihood function: -3.3738885624
Iteration: 20 Log-likelihood function: -3.3423876581
Iteration: 21 Log-likelihood function: -3.31319343769
Iteration: 22 Log-likelihood function: -3.2860493346
Iteration: 23 Log-likelihood function: -3.2607366863
```

3 Using the logistic regression model for classification

Classification is performed by calling the `classify`

