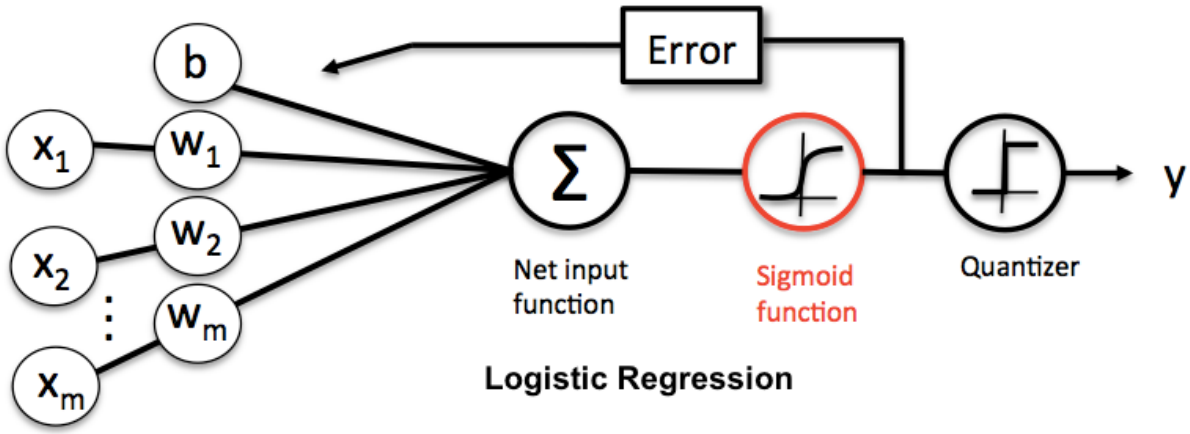
Machine Learning FAQ

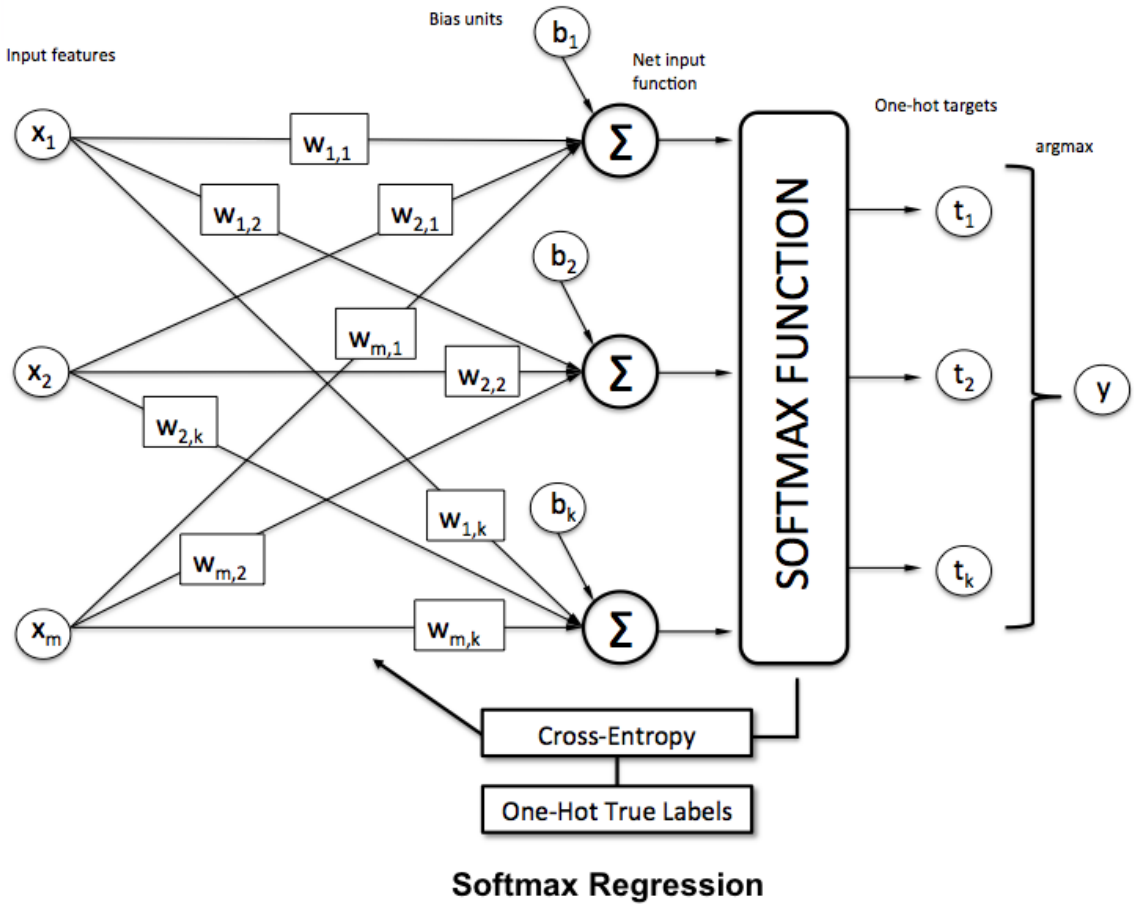
Index

What is Softmax regression and how is it related to Logistic regression?

Softmax Regression (synonyms: Multinomial Logistic, Maximum Entropy Classifier, or just Multi-class Logistic Regression) is a generalization of logistic regression that we can use for multi-class classification (under the assumption that the classes are mutually exclusive). In contrast, we use the (standard) Logistic Regression model in binary classification tasks.

Now, let me briefly explain how that works and how softmax regression differs from logistic regression. I have a more detailed explanation on logistic regression here: LogisticRegression - mlxtend , but let me re-use one of the figures to make things more clear:





As the name suggests, in softmax regression (SMR), we replace the sigmoid logistic function by the so-called *softmax function* ϕ:

$$P(y = j \mid z^{(i)}) = \phi_{softmax}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=0}^{k} e^{z_k^{(i)}}},$$

where we define the net input z as

$$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \sum_{l=0}^{m} w_l x_l = \mathbf{w}^T \mathbf{x}.$$

(*w* is the weight vector, *x* is the feature vector of 1 training sample, and w_0 is the bias unit.) Now, this softmax function computes the probability that this training sample $x^{(i)}$ belongs to class *j* given the weight and net input $z^{(i)}$. So, we compute the probability $p(y = j \mid x^{(i)}; w_j)$ for each class label in $j = 1, \ldots, k$. Note the normalization term in the denominator which causes these class probabilities to sum up to one.

To illustrate the concept of softmax, let us walk through a concrete example. Let's assume we have a training set consisting of 4 samples from 3 different classes (0, 1, and 2).

- $x_0 \rightarrow$ class 0
- $x_1 \rightarrow$ class 1
- $x_2 \rightarrow$ class 2
- $x_3 \rightarrow$ class 2

First, we want to encode the class labels into a format that we can more easily work with; we apply one-hot encoding:

```
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 0.  0.  1.]]
```

A sample that belongs to class 0 (the first row) has a 1 in the first cell, a sample that belongs to class 2 has a 1 in the second cell of its row, and so forth. Next, let us define the feature matrix of our 4 training samples. Here, we assume that our dataset consists of 2 features; thus, we create a 4×(2+1) dimensional matrix (+1 one for the bias term).

```
Inputs X:
 [[ 0.1  0.5]
 [ 1.1  2.3]
 [-1.1 -2.3]
 [-1.5 -2.5]]

Weights W:
 [[ 0.1  0.2  0.3]
 [ 0.1  0.2  0.3]]

bias:
 [ 0.01  0.1   0.1 ]

X with "ones":
 [[ 1.   0.1  0.5]
 [ 1.   1.1  2.3]
 [ 1.  -1.1 -2.3]
 [ 1.  -1.5 -2.5]]

W with bias:
 [[ 0.01  0.1   0.1 ]
 [ 0.1   0.2   0.3 ]
 [ 0.1   0.2   0.3 ]]
```

Similarly, we created a (2+1)×3 dimensional weight matrix (one row per feature and one column for each class).

To compute the net input, we multiply the 4×(2+1) feature matrix **X** with the (2+1)×3 (n_features × n_classes) weight matrix **W**.

Z = WX

which yields a 4×3 output matrix (n_samples × n_classes).

```
net input:
 [[ 0.07  0.22  0.28]
 [ 0.35  0.78  1.12]
 [-0.33 -0.58 -0.92]
 [-0.39 -0.7  -1.1 ]]
```

Now, it's time to compute the softmax activation that we discussed earlier:

$$P(y = j \mid z^{(i)}) = \phi_{softmax}(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{j=0}^{k} e^{z_k^{(i)}}}.$$

```
softmax:
 [[ 0.29450637  0.34216758  0.36332605]
 [ 0.21290077  0.32728332  0.45981591]
 [ 0.42860913  0.33380113  0.23758974]
 [ 0.44941979  0.32962558  0.22095463]]
```

As we can see, the values for each sample (row) nicely sum up to 1 now. E.g., we can say that the first sample
`[ 0.29450637 0.34216758 0.36332605]` has a 29.45% probability to belong to class 0. Now, in order to turn these probabilities back into class labels, we could simply take the argmax-index position of each row:

```
predicted class labels:  [2 2 0 0]
```

As we can see, our predictions are terribly wrong, since the correct class labels are `[0, 1, 2, 2]`. Now, in order to train our logistic model (e.g., via an optimization algorithm such as gradient descent), we need to define a cost function *J* that we want to minimize:

$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=0}^{n} H(T_i, O_i)$$

which is the average of all cross-entropies over our n training samples. The cross-entropy function is defined as

$$H(T_i, O_i) = -\sum_m T_i \cdot log(O_i).$$

Here the T stands for "target" (the true class labels) and the O stands for output (the computed probability via softmax; ***not*** the predicted class label).

```
Cross Entropy: [ 1.22245465  1.11692907  1.43720989  1.50979788]
```

In order to learn our softmax model via gradient descent, we need to compute the derivative

$$\nabla \mathbf{w}_j J(\mathbf{W}),$$

which we then use to update the weights in opposite direction of the gradient:

$$\mathbf{w}_j := \mathbf{w}_j - \alpha \nabla \mathbf{w}_j J(\mathbf{W})$$

for each class j.

$$j \in \{0, 1, \ldots, k\}.$$

(Note that w_j is the weight vector for the class *y=j*.) I don't want to walk through more tedious details here, but this cost derivative turns out to be simply:

$$\nabla \mathbf{w}_j J(\mathbf{W}) = -\frac{1}{n} \sum_{i=0}^{n} \left[\mathbf{x}^{(i)} \left(T_i - O_i \right) \right]$$

Using this cost gradient, we iteratively update the weight matrix until we reach a specified number of epochs (passes over the training set) or reach the desired cost threshold.