Multiclass Logistic Regression, Multilayer Perceptron

Milan Straka

October 26, 2020
Logistic Regression

An extension of perceptron, which models the conditional probabilities of $p(C_0 | \mathbf{x})$ and of $p(C_1 | \mathbf{x})$. Logistic regression can in fact handle also more than two classes, which we will see shortly.

Logistic regression employs the following parametrization of the conditional class probabilities:

$$p(C_1 | \mathbf{x}) = \sigma(\mathbf{x}^T \mathbf{w} + b)$$
$$p(C_0 | \mathbf{x}) = 1 - p(C_1 | \mathbf{x}),$$

where $\sigma$ is a sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

It can be trained using the SGD algorithm.
We denote the output of the “linear part” of the logistic regression as \( \tilde{y}(x; w) = x^T w \) and the overall prediction as \( y(x; w) = \sigma(\tilde{y}(x; w)) = \sigma(x^T w). \)

The logistic regression output \( y(x; w) \) models the probability of class \( C_1, p(C_1|x) \).

To give some meaning to the output of the linear part \( \tilde{y}(x; w) \), starting with

\[
p(C_1|x) = \sigma(\tilde{y}(x; w)) = \frac{1}{1 + e^{-\tilde{y}(x; w)}},
\]

we arrive at

\[
\tilde{y}(x; w) = \log \left( \frac{p(C_1|x)}{1 - p(C_1|x)} \right) = \log \left( \frac{p(C_1|x)}{p(C_0|x)} \right),
\]

which is called a logit and it is a logarithm of odds of the probabilities of the two classes.
Logistic Regression

To train the logistic regression, we use MLE (the maximum likelihood estimation). Its application is straightforward, given that $p(C_1 | \mathbf{x}; \mathbf{w})$ is directly the model output $y(\mathbf{x}; y)$.

Therefore, the loss for a batch $\mathbf{X} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots, (\mathbf{x}_N, t_N)\}$ is

$$
\mathcal{L}(\mathbf{X}) = \frac{1}{N} \sum_i - \log(p(C_{t_i} | \mathbf{x}_i; \mathbf{w})).
$$

**Input:** Input dataset ($\mathbf{X} \in \mathbb{R}^{N \times D}$, $\mathbf{t} \in \{0, +1\}^N$), learning rate $\alpha \in \mathbb{R}^+$.  

- $\mathbf{w} \leftarrow 0$
- until convergence (or until patience is over), process batch of $N$ examples:
  - $\mathbf{g} \leftarrow \frac{1}{N} \sum_i \nabla \mathbf{w} - \log(p(C_{t_i} | \mathbf{x}_i; \mathbf{w}))$
  - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \mathbf{g}$
Linearity in Logistic Regression

Figure 4.12 of Pattern Recognition and Machine Learning.
Generalized Linear Models

The logistic regression is in fact an extended linear regression. A linear regression model, which is followed by some activation function $a$, is called generalized linear model:

$$p(t|x; w, b) = y(x; w, b) = a(\bar{y}(x; w, b)) = a(x^T w + b).$$

<table>
<thead>
<tr>
<th>Name</th>
<th>Activation</th>
<th>Distribution</th>
<th>Loss</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear regression</td>
<td>identity</td>
<td>$?$</td>
<td>$\text{MSE} \propto E(y(x) - t)^2$</td>
<td>$(y(x) - t)x$</td>
</tr>
<tr>
<td>logistic regression</td>
<td>$\sigma(y)$</td>
<td>Bernoulli</td>
<td>$\text{NLL} \propto E - \log(p(t</td>
<td>x))$</td>
</tr>
</tbody>
</table>
During regression, we predict a number, not a real probability distribution. In order to generate a distribution, we might consider a distribution with the mean of the predicted value and a fixed variance $\sigma^2$ – the most general such a distribution is the normal distribution.
Mean Square Error as MLE

Therefore, assume our model generates a distribution \( p(t|x; w) = \mathcal{N}(t; y(x; w), \sigma^2) \).

Now we can apply MLE and get

\[
\arg \max_w p(t|X; w) = \arg \min_w \sum_{i=1}^N - \log p(t_i|x_i; w)
\]

\[
= \arg \min_w \sum_{i=1}^N \log \sqrt{\frac{1}{2\pi\sigma^2}} e^{-\frac{(t_i-y(x_i;w))^2}{2\sigma^2}}
\]

\[
= \arg \min_w -N \log(2\pi\sigma^2)^{-1/2} - \sum_{i=1}^N -\frac{(t_i - y(x_i;w))^2}{2\sigma^2}
\]

\[
= \arg \min_w \sum_{i=1}^N \frac{(t_i - y(x_i;w))^2}{2\sigma^2} = \arg \min_w \frac{1}{N} \sum_{i=1}^N (t_i - y(x_i;w))^2.
\]
Generalized Linear Models

We have therefore extended the GLM table to

<table>
<thead>
<tr>
<th>Name</th>
<th>Activation</th>
<th>Distribution</th>
<th>Loss</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear regression</td>
<td>identity</td>
<td>Normal</td>
<td>NLL $\propto$ MSE</td>
<td>$(y(x) - t)x$</td>
</tr>
<tr>
<td>logistic regression</td>
<td>$\sigma(y)$</td>
<td>Bernoulli</td>
<td>NLL $\propto E - \log(p(t</td>
<td>x))$</td>
</tr>
</tbody>
</table>
Multiclass Logistic Regression

To extend the binary logistic regression to a multiclass case with $K$ classes, we:

- generate $K$ outputs, each with its own set of weights, so that for $W \in \mathbb{R}^{D \times K}$,

  \[
  \tilde{y}(x; W) = x^T W, \quad \text{or in other words, } \tilde{y}(x; W)_i = x^T (W_{*,i})
  \]

- generalize the sigmoid function to a softmax function, such that

  \[
  \text{softmax}(y)_i = \frac{e^{y_i}}{\sum_j e^{y_j}}.
  \]

Note that the original sigmoid function can be written as

\[
\sigma(x) = \text{softmax } ([x \ 0])_0 = \frac{e^x}{e^x + e^0} = \frac{1}{1 + e^{-x}}.
\]

The resulting classifier is also known as multinomial logistic regression, maximum entropy classifier or softmax regression.
Using the softmax function, we naturally define that

\[ p(C_i | \mathbf{x}; \mathbf{W}) = y(\mathbf{x}; \mathbf{W})_i = \text{softmax}(\tilde{y}(\mathbf{x}; \mathbf{W}))_i = \text{softmax}(\mathbf{x}^T \mathbf{W})_i = \frac{e^{(\mathbf{x}^T \mathbf{W})_i}}{\sum_j e^{(\mathbf{x}^T \mathbf{W})_j}}. \]

Considering the definition of the softmax function, it is natural to obtain the interpretation of the linear part of the model \( \tilde{y}(\mathbf{x}; \mathbf{W}) \) as logits:

\[ \tilde{y}(\mathbf{x}; \mathbf{W})_i = \log(p(C_i | \mathbf{x}; \mathbf{W})) + c. \]

The constant \( c \) is present, because the output of the model is overparametrized (the probability of for example the last class could be computed from the remaining ones). This is connected to the fact that softmax is invariant to addition of a constant:

\[ \text{softmax}(\mathbf{y} + c)_i = \frac{e^{y_i + c}}{\sum_j e^{y_j + c}} = \frac{e^{y_i}}{\sum_j e^{y_j}} \cdot \frac{e^c}{e^c} = \text{softmax}(\mathbf{y})_i. \]
Multiclass Logistic Regression

The difference between softmax and sigmoid output can be compared on the binary case, where the binary logistic regression outputs of the linear part of the model are

\[ \bar{y}(x; w) = \log \left( \frac{p(C_1|x; w)}{p(C_0|x; w)} \right), \]

while the outputs of the softmax variant with two outputs can be interpreted as

\[ \bar{y}(x; W)_0 = \log(p(C_0|x; W)) + c \quad \text{and} \quad \bar{y}(x; W)_1 = \log(p(C_1|x; W)) + c. \]

If we consider \( \bar{y}(x; W)_0 \) to be zero, the model can then predict only the probability \( p(C_1|x) \), and the constant \( c \) is fixed to \(-\log(p(C_0|x; W))\), recovering the original interpretation.

Generalizing to a \( K \)-class classification, we could produce only \( K - 1 \) outputs and define \( \bar{y}_K = 0 \), resulting in the interpretation of the linear part outputs analogous to the binary case:

\[ \bar{y}(x; W)_i = \log \left( \frac{p(C_i|x; W)}{p(C_K|x; W)} \right). \]
Multiclass Logistic Regression

To train $K$-class classification, analogously to the binary logistic regression we can use MLE and train the model using minibatch stochastic gradient descent:

**Input:** Input dataset $(\mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{t} \in \{0, 1, \ldots, K - 1\}^N)$, learning rate $\alpha \in \mathbb{R}^+$.  

**Model:** Let $\mathbf{w}$ denote all parameters of the model (in our case, the parameters are a weight matrix $\mathbf{W}$ and maybe a bias vector $\mathbf{b}$).

- $\mathbf{w} \leftarrow 0$
- until convergence (or until patience is over), process batch of $N$ examples:
  - $\mathbf{g} \leftarrow \frac{1}{N} \sum_i \nabla_\mathbf{w} \log(p(C_{t_i} | \mathbf{x}_i; \mathbf{w}))$
  - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \mathbf{g}$
Note that the decision regions of the binary/multiclass logistic regression are convex (and therefore connected). To see this, consider $x_A$ and $x_B$ in the same decision region $R_k$.

Any point $x$ lying on the line connecting them is their linear combination, $x = \lambda x_A + (1 - \lambda) x_B$, and from the linearity of $\tilde{y}(x) = x^T W$ it follows that

$$\tilde{y}(x) = \lambda \tilde{y}(x_A) + (1 - \lambda) \tilde{y}(x_B).$$

Given that $\tilde{y}(x_A)_k$ was the largest among $\tilde{y}(x_A)$ and also given that $\tilde{y}(x_B)_k$ was the largest among $\tilde{y}(x_B)$, it must be the case that $\tilde{y}(x)_k$ is the largest among all $\tilde{y}(x)$.
The multiclass logistic regression can now be added to the GLM table:

<table>
<thead>
<tr>
<th>Name</th>
<th>Activation</th>
<th>Distribution</th>
<th>Loss</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear regression</td>
<td>identity</td>
<td>Normal</td>
<td>NLL $\propto$ MSE</td>
<td>$(y(x) - t)x$</td>
</tr>
<tr>
<td>logistic regression</td>
<td>$\sigma(y)$</td>
<td>Bernoulli</td>
<td>NLL $\propto$ $\mathbb{E} - \log(p(t</td>
<td>x))$</td>
</tr>
<tr>
<td>multiclass logistic regression</td>
<td>softmax($y$)</td>
<td>categorical</td>
<td>NLL $\propto$ $\mathbb{E} - \log(p(t</td>
<td>x))$</td>
</tr>
</tbody>
</table>

NPFL129, Lecture 4
There exist several others GLMs, we now describe a last one, this time for regression and not for classification. Compared to regular linear regression, where we assume the output distribution is normal, we turn our attention to Poisson distribution.

### Poisson Distribution

Poisson distribution is a discrete distribution suitable for modeling the probability of a given number of events occurring in a fixed time interval, if these events occur with a known rate and independently of each other.

\[
P(x = k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}
\]

It is easy to show that if \( x \) has Poisson distribution,

\[
\mathbb{E}[x] = \lambda \\
\text{Var}(x) = \lambda
\]
Poisson Distribution

An important difference to normal distribution is that the latter assumes the variance does not depend on the mean, i.e., that the model “makes errors of the same magnitude everywhere”. On the other hand, the variance of a Poisson distribution increases with the mean.
Poisson regression is a generalized linear model producing a Poisson distribution (i.e., the mean rate $\lambda$).

Again, we use NLL as the loss. To choose a suitable activation, we might be interested in obtaining the same gradient as for other GLMs – solving for an activation function while requiring the gradient to be $(a(\bar{y}(x)) - t) \cdot x$ yields $a(y) = \exp(y)$, which means the linear part of the model is predicting $\log(\lambda)$.

<table>
<thead>
<tr>
<th>Name</th>
<th>Activation</th>
<th>Distribution</th>
<th>Loss</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear regression</td>
<td>identity</td>
<td>Normal</td>
<td>NLL $\propto$ MSE</td>
<td>$(y(x) - t) \cdot x$</td>
</tr>
<tr>
<td>logistic regression</td>
<td>$\sigma(y)$</td>
<td>Bernoulli</td>
<td>NLL $\propto$ $E - \log(p(t</td>
<td>x))$</td>
</tr>
<tr>
<td>multiclass logistic regression</td>
<td>$\text{softmax}(y)$</td>
<td>categorical</td>
<td>NLL $\propto$ $E - \log(p(t</td>
<td>x))$</td>
</tr>
<tr>
<td>Poisson regression</td>
<td>$\exp(y)$</td>
<td>Poisson</td>
<td>NLL $\propto$ $E - \log(p(t</td>
<td>x))$</td>
</tr>
</tbody>
</table>
We can reformulate the generalized linear models in the following framework.

- Assume we have an input node for every input feature.
- Additionally, we have an output node for every model output (one for linear regression or binary classification, $K$ for classification in $K$ classes).
- Every input node and output node are connected with an directed edge, and every edge has an associated weight.
- Value of every (output) node is computed by summing the values of predecessors multiplied by the corresponding weights, added to a bias of this node, and finally passed through an activation function $a$:

$$y_i = a \left( \sum_j x_j w_{j,i} + b_i \right)$$

or in matrix form $\mathbf{y} = a(\mathbf{x}^T \mathbf{W} + \mathbf{b})$ or, for a batch of examples $\mathbf{X}$, $\mathbf{Y} = a(\mathbf{XW} + \mathbf{b})$. 
Multilayer Perceptron

We now extend the model by adding a **hidden layer** with activation $f$.

- The computation is performed analogically:

$$h_i = f \left( \sum_j x_j w_{j,i}^h + b_i^h \right),$$

$$y_i = a \left( \sum_j h_j w_{j,i}^y + b_i^y \right),$$

or in matrix form

$$h = f \left( x^T W^h + b^h \right),$$

$$y = a \left( h^T W^y + b^y \right),$$

and for batch of inputs $H = f \left( X W^h + b^h \right)$ and $Y = a \left( H W^y + b^y \right)$. 
Multilayer Perceptron

Note that:

- the structure of the *input* layer depends on the input features;
- the structure and the *activation* function of the *output* layer depends on the target data;
- however, the *hidden* layer has no pre-image in the data and is completely arbitrary – which is the reason why it is called a *hidden* layer.

Also note that we can absorb biases into weights analogously to the generalized linear models.
Output Layer Activation Functions

- **regression:**
  - identity activation: we model normal distribution on output (linear regression)
  - $\exp(x)$: we model Poisson distribution on output (Poisson regression)

- **binary classification:**
  - $\sigma(x)$: we model the Bernoulli distribution (the model predicts a probability)

$$
\sigma(x) \overset{\text{def}}{=} \frac{1}{1 + e^{-x}}
$$

- **$K$-class classification:**
  - $\text{softmax}(\mathbf{x})$: we model the (usually overparametrized) categorical distribution

$$
\text{softmax}(\mathbf{x}) \propto e^{\mathbf{x}}, \quad \text{softmax}(\mathbf{x})_i \overset{\text{def}}{=} \frac{e^{x_i}}{\sum_j e^{x_j}}
$$
Hidden Layer Activation Functions

- no activation (identity): does not help, composition of linear mapping is a linear mapping
- $\sigma$ (but works suboptimally – nonsymmetrical, $\frac{d\sigma}{dx}(0) = 1/4$)
- $\tanh$
  - result of making $\sigma$ symmetrical and making derivation in zero 1
  - $\tanh(x) = 2\sigma(2x) - 1$
- ReLU
  - $\max(0, x)$
  - the most common non-linear activation used nowadays

From Sigmoid Function $\sigma(x)$ to Tanh
The multilayer perceptron can be trained using again a minibatch SGD algorithm:

**Input:** Input dataset \( (\mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{t} \text{ targets}) \), learning rate \( \alpha \in \mathbb{R}^+ \).

**Model:** Let \( \mathbf{w} \) denote all parameters of the model (all weight matrices and bias vectors).

- initialize \( \mathbf{w} \)
  - set weights randomly, for example in \( U\left[ -1/\sqrt{M}, 1/\sqrt{M} \right] \) range, where \( M \) is the size of the layer computed by the corresponding edge
  - set biases to 0
- until convergence (or until patience is over), process batch of \( N \) examples:
  - \( \mathbf{g} \leftarrow \nabla_{\mathbf{w}} \frac{1}{N} \sum_j - \log p(t_j | \mathbf{x}_j; \mathbf{w}) \)
  - \( \mathbf{w} \leftarrow \mathbf{w} - \alpha \mathbf{g} \)
Assume we have an MLP with input of size $D$, weights $W^h \in \mathbb{R}^{D \times H}$, $b^h \in \mathbb{R}^H$, hidden layer of size $H$ and activation $f$ with weights $W^y \in \mathbb{R}^{H \times K}$, $b^y \in \mathbb{R}^K$, and finally an output layer of size $K$ with activation $a$.

In order to compute the gradient of the loss $L$ with respect to all weights, you should proceed gradually:

- first compute $\frac{\partial L}{\partial y}$,
- then compute $\frac{\partial y}{\partial y_{in}}$, where $y_{in}$ are the inputs to the output layer (i.e., before applying activation function $a$; in other words, $y = a(y_{in})$),
- then compute $\frac{\partial y_{in}}{\partial W^y}$ and $\frac{\partial y_{in}}{\partial b^y}$, which allows us to obtain $\frac{\partial L}{\partial W^y} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial y_{in}} \cdot \frac{\partial y_{in}}{\partial W^y}$ and analogously $\frac{\partial L}{\partial b^y}$,
- followed by $\frac{\partial y_{in}}{\partial h}$ and $\frac{\partial h}{\partial h_{in}}$,
- and finally using $\frac{\partial h_{in}}{\partial W^h}$ and $\frac{\partial h_{in}}{\partial b^h}$ to compute $\frac{\partial L}{\partial W^h}$ and $\frac{\partial L}{\partial b^h}$. 


Hidden Layer Interpretation and Initialization

One way how to interpret the hidden layer is:

- the part from the hidden layer to the output layer is the previously used generalized linear model (linear regression, logistic regression, ...);
- the part from the inputs to the hidden layer can be considered automatically constructed features. The features are a linear mapping of the input values followed by a non-linearity, and the theorem on the next slide proves they can always be constructed to achieve as good fit of the training data as required.

Note that the weights in an MLP must be initialized randomly. If we used just zeros, all the constructed features (hidden layer nodes) would behave identically and we would never distinguish them.

Using random weights corresponds to using random features, which allows the SGD to make progress (improve the individual features).
Universal Approximation Theorem '89

Let $\varphi(x)$ be a nonconstant, bounded and nondecreasing continuous function. (Later a proof was given also for $\varphi = \text{ReLU}$.)

Then for any $\varepsilon > 0$ and any continuous function $f$ on $[0, 1]^m$ there exists an $N \in \mathbb{N}, v_i \in \mathbb{R}, b_i \in \mathbb{R}$ and $w_i \in \mathbb{R}^m$, such that if we denote

$$F(x) = \sum_{i=1}^{N} v_i \varphi(x^T w_i + b_i),$$

then for all $x \in [0, 1]^m$:

$$|F(x) - f(x)| < \varepsilon.$$
Universal Approximation Theorem for ReLUs

Sketch of the proof:

• If a function is continuous on a closed interval, it can be approximated by a sequence of lines to arbitrary precision.

\[ n_1(x) = \text{ReLU}(-5x - 7.7) \]
\[ n_2(x) = \text{ReLU}(-1.2x - 1.3) \]
\[ n_3(x) = \text{ReLU}(1.2x + 1) \]
\[ n_4(x) = \text{ReLU}(1.2x - .2) \]
\[ n_5(x) = \text{ReLU}(2x - 1.1) \]
\[ n_6(x) = \text{ReLU}(3x - 5) \]

\[ Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x) \]

• However, we can create a sequence of $k$ linear segments as a sum of $k$ ReLU units – on every endpoint a new ReLU starts (i.e., the input ReLU value is zero at the endpoint), with a tangent which is the difference between the target tangent and the tangent of the approximation until this point.

https://miro.medium.com/max/844/1*iibPNQgI7oKjpGsmzPDKw.png
Universal Approximation Theorem for Squashes

Sketch of the proof for a squashing function $\varphi(x)$ (i.e., nonconstant, bounded and nondecreasing continuous function like sigmoid):

- We can prove $\varphi$ can be arbitrarily close to a hard threshold by compressing it horizontally.

$$y = \frac{1}{1 + e^{-(w^T x + b)}}$$

- Then we approximate the original function using a series of straight line segments.