GHENT
UNIVERSITY

Lecture Notes

E016350: Artificial Intelligence

Multiclass Logistic Regression

Aleksandra Pizurica

Spring 2024

Contents

Disclaimer*: These lecture notes were written by Prof. Aleksandra Pizurica to accompany the slides of the course E016350: Artificial Intelligence, facilitating their understanding. The lecture notes are not meant to be self-contained, and do not cover all the study material in the course. They are by no means meant to replace the recommended textbook and do not necessarily cover all the relevant aspects that are presented in the slides and explained in the lectures. Some sections are adapted from the book of S. Russel and P. Norvig: Artificial Intelligence: A Modern Approach.*

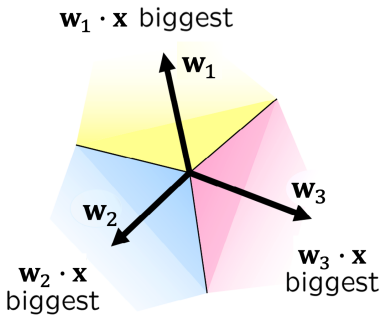

Figure 1: The concept of multiclass linear classification illustrated on a case with three classes. The input data point $\mathbf{x}$ is assigned to the class that gives the biggest score. Credit: D. Klein & P. Abbeel [1].

1 Multiclass linear classification

So far we considered only binary linear classification. Now we turn to a more general case where we can have more than two classes. For example, we want to predict the value of a handwritten digit or to classify newspaper articles into categories culture, science, sports, politics etc. We still want to define the decision boundaries based on **linear functions** of the input, i.e., based on linear combinations of input features. This task is called **multiclass linear classification**.

Let our input be a d-dimensional vector as before $\mathbf{x} \in \mathbb{R}^d$. We now have a weight vector $\mathbf{w}_y \in \mathbb{R}^d$ for each output class $y \in \{1, \ldots, K\}$, and a new input $\mathbf{x}$ is classified based on the **scores** $\mathbf{w}_y \cdot \mathbf{x}$ that are computed for every class. We will put all the weight vectors together in one long vector $\mathbf{w} = [(\mathbf{w}_1)^\top, \ldots, (\mathbf{w}_K)^\top]^\top \in \mathbb{R}^{Kd}$ and we'll denote the prediction same as before by $h_{\mathbf{w}}(\mathbf{x})$.

1.1 Multiclass perceptron

Given the setup above, the prediction rule "the highest score wins":

$$h_{\mathbf{w}}(\mathbf{x}) = \arg\max_i \ \mathbf{w}_i \cdot \mathbf{x} \tag{1}$$

extends directly the linear binary classification with a **hard threshold** to multiple classes. This classification approach is illustrated in Fig. 1 and is often referred to as the **multiclass perceptron**. The scores $z_i = \mathbf{w}_i \cdot \mathbf{x}$ are also called **activations** (this is the terminology that we will use commonly with neural networks).

Often it is convenient to represent multiclass classification with **one-hot encoding**. This means that the target output (the correct classification result) is represented as a vector $\mathbf{t}$ with all zeroes except one entry "1", which indicates the correct class. For example, if the correct class out of K possibles classes is the k-th class, the one-hot encoded target output is

$$\mathbf{t} = \underbrace{[0, \ldots, 0, 1, 0, \ldots, 0]^\top}_{\text{entry } k \text{ is one}} \in \mathbb{R}^K$$

We can represent the multiclass perceptron prediction with one-hot encoding as follows. Let $\mathbf{o} \in \mathbb{R}^K$ be the one-hot encoded output vector. Then for the multiclass perceptron

$$o_k = \begin{cases} 1 & \text{if } k = \arg\max_i \ \mathbf{w}_i \cdot \mathbf{x} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

1.2 Multiclass logistic regression and the softmax rule

The question now is how to turn the *hard* multiclass classification that we defined above into a soft one. In the binary case we replaced the hard threshold with the sigmoid function and we called the resulting model logistic regression. The nice property of the sigmoid (logistic) function was that it provided a probabilistic interpretation of the output as the probability of belonging to class "1".

We can equivalently turn the scores for multiclass classification into probabilities for belonging to the corresponding classes by using the **softmax** rule:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}, \quad i = 1, \ldots, K \tag{3}$$

The original activations z_i are transformed this way to **softmax activations**. The resulting approach is **multiclass logistic regression** (also called **multinomial logistic regression** or **softmax regression**) where the hypothesis is defined as:

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{bmatrix} P(y=1|\mathbf{x},\mathbf{w}) \\ P(y=2|\mathbf{x},\mathbf{w}) \\ \vdots \\ P(y=K|\mathbf{x},\mathbf{w}) \end{bmatrix} = \frac{1}{\sum_{k=1}^{K} e^{\mathbf{w}_k \cdot \mathbf{x}}} \begin{bmatrix} e^{\mathbf{w}_1 \cdot \mathbf{x}} \\ e^{\mathbf{w}_2 \cdot \mathbf{x}} \\ \vdots \\ e^{\mathbf{w}_K \cdot \mathbf{x}} \end{bmatrix} \tag{4}$$

where $\mathbf{w} = [(\mathbf{w}_1)^\top, \ldots, (\mathbf{w}_K)^\top]^\top$. If we denote the output vector by $\mathbf{o} = h_{\mathbf{w}}(\mathbf{x})$ we can write

$$o_k = softmax(\mathbf{w}_k \cdot \mathbf{x}), \quad k = 1, \ldots, K \tag{5}$$

Note how this "softens" the hard classification rule in Eq (2).

2 Learning weights for multiclass linear classification

Let us now see how we learn the weights from the training data for the two above presented multiclass classification methods.

2.1 Multiclass perceptron learning rule

Remember the perceptron learning rule for binary classification with $y \in \{0, 1\}$, which can be written in a vector form as $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y - h_{\mathbf{w}}(\mathbf{x}))\mathbf{x}$. It did nothing if the output was correct, and otherwise the weights were either increased or decreased by $\alpha\mathbf{x}$ to nudge them in the right direction (increasing if $y = 1$ and $h_{\mathbf{w}}(\mathbf{x}) = 0$ and decreasing in $y = 0$ and $h_{\mathbf{w}}(\mathbf{x}) = 1$). This is simply extended to the case with multiple classes as follows:

- If $h_{\mathbf{w}}(\mathbf{x}) = y$ do nothing
- If $h_{\mathbf{w}}(\mathbf{x}) \neq y$ update the weights for the true class y and for the predicted class $y^* = h_{\mathbf{w}}(\mathbf{x})$
  - Update the **correct** class vector as $\mathbf{w}_y \leftarrow \mathbf{w}_y + \alpha\mathbf{x}$
  - Update the **wrong** class vector as $\mathbf{w}_{y^*} \leftarrow \mathbf{w}_{y^*} - \alpha\mathbf{x}$
  - Do **not** change the weights of any other class

2.2 Optimization for multiclass logistic regression

For multiclass logistic regression we optimize the weights similarly as we did with the logistic regression in the binary case: by maximizing the likelihood of the weights given the training data:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

Assuming as before that the training examples were generated independently, the likelihood is:

$$\mathcal{L}(\mathbf{w}) = \prod_{i=1}^{N} P(y^{(i)}|\mathbf{x}^{(i)}, \underbrace{\mathbf{w}_1, \ldots, \mathbf{w}_K}_{\mathbf{w}}) = \prod_{i=1}^{N} P(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) \tag{6}$$

where

$$P(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = \frac{e^{\mathbf{w}_{y^{(i)}} \cdot \mathbf{x}^{(i)}}}{\sum_y e^{\mathbf{w}_{y^{(i)}} \cdot \mathbf{x}^{(i)}}}$$

Again, as was the case with the binary logistic regression, we will perform the desired optimization easier on =the logarithm of the likelihood:

$$\ell(\mathbf{w}) = \log \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} \log P(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) \tag{7}$$

The optimization objective is now equivalently expressed as maximizing the likelihood or minimizing the negative log-likelihood, i.e., the training loss is now the negative log-likelihood and we have that:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \; -\ell(\mathbf{w}) = \arg\min_{\mathbf{w}} \; -\sum_{i=1}^{N} \log P(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) \tag{8}$$

Thus the update rule with the stochastic gradient descent is

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{i=1}^{N} \nabla \log P(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) \tag{9}$$

Additional insight (optional reading): It is possible to express this update rule analytically and to show that it is a direct extension of the update rule for the weights in the case of binary logistic regression. Let $\mathbf{t}^{(i)}$ and $\mathbf{o}^{(i)}$ denote **one-hot** encoded target and predicted output for the ith example. The update rule for multiclass logistic regression is:

$$\mathbf{w}_k \leftarrow \mathbf{w}_k + \alpha \sum_{i=1}^{N} (t_k^{(i)} - o_k^{(i)}))\mathbf{x}^{(i)}, \qquad k = 1, \ldots, K$$

The log-likelihood loss in the logistic regression, which is often called the **logistic loss** or just **log loss**) is in the literature often called also **cross-entropy loss** (although strictly speaking the logistic loss is an approximation of the true cross-entropy loss, which would require the actual (unknown) distribution of the examples, and we are approximating this unknown distribution by its samples contained in the training set). Nevertheless, these terms are now often used interchangeably and the term cross-entropy loss is common in the machine learning community.

References

[1] D. Klein and P. Abbeel. *Intro to AI (ai.berkeley.edu)*. UC Berkeley, 2023.