# Logistic Regression

Urša Zrimšek

*uz2273@student.uni-lj.si, 63200441*

## Introduction

This is the report of second homework for class Machine Learning for Data Science 1, in which we implemented multinomial and ordinal logistic regression.

The models are applied to `dataset.csv` dataset, that is composed of 250 rows with 12 numeric and categorical columns, first of them being an answer to the questionnaire, with answers from very poor to very good (5 classes, changed in preprocessing to 0 - 4), and latter ones describe other available information that describe students that responded - their age, sex (we changed M and F to 0 and 1), year of study and grades for multiple exams.

## Models

For both models, we implemented the log-likelihood of the model and fitted it using maximum likelihood estimation. For optimization we used `fmin_l_bfgs_b` from the scipy.optimize library. With it we minimized the negative value of log-likelihood. We didn't calculate the gradient, but we used its numerical approximation. Let us now derive the likelihoods.

### Multinomial logistic regression

MLR is a generalization of logistic regression to more than 2 objects. Let's say we are choosing from $m$ objects. We then calculate latent strengths for $i \in \{1,\dots,m-1\}$: $u_{i,j} = \beta_j^T x_i + \beta_0$, $x_i$ being the input variables of one observation and $\beta_j$ being the weights for each (except for the last) object. Our last object is the reference category, meaning $u_{im} = 0$.

We added intercept $\beta_0$, because of scaling the data at the beginning - we subtracted the mean of every column of the training data, and divided it by its standard deviation. We used the same scaling also on test data (with mean and std of training data).

The probabilities of choosing $j$-th object in $i$-th observation are equal to:

$$P(\text{choose object j in i-th}) = \frac{e^{u_{ij}}}{1+\sum_{i=1}^{m-1} e^{u_{ij}}}. \quad (1)$$

Since our goal is to optimize the weights, we should write the log-likelihood in a notation that uses them. And as we already know, log likelihood is defined as $l(\beta) = \log(L(\beta)) = \log(\Pi_i \Pi_c (P(y_i|\beta) I(y_i = c))) = \sum_i \sum_c (\log(P(y_i|\beta) I(y_i = c)))$. $I(y_i = c)$ is an indicator that i-th observation belongs to class c. It simplifies our log likelihood to:

$$l(\beta) = \sum_i \log(\frac{e^{\beta_{y_i} x_i}}{\sum_{j=1}^{m-1} e^{\beta_j x_i} + 1}),$$

where $y_i$ is the correct class of i-th observation. To find $\beta$-s at which the likelihood has its maximum value, we input the negative value of the likelihood, together with initial guess with all elements being 1 and our training data into previously mentioned function.

For prediction of new values, we calculated probabilities for all objects - equation 1, and returned the object that corresponds to the maximum probability, or, if the argument `return_prob` was set to `True`, we returned all probabilities (needed for log loss calculations).

### Ordinal logistic regression

At OLR we will optimize parameters $\beta$ (vector of the same length as the input data of one observation), intercept $\beta_0$ (again because of standardization of data) and thresholds $\{-\infty, 0, t_i, i \in \{2,\dots,m-1\}, \infty\}$. Since we need our thresholds to be ordered, we will instead of them, optimize the distances - $\delta_i, i \in \{1,\dots,m-1\}$ between them (so that we can easily add a constraint $\delta_i > \varepsilon > 0$ to `fmin_l_bfgs_b`).

Here the probabilities of choosing $j$-th object in $i$-th observation are equal to:

$$p_i(j) = P(\text{choose j in i-th}) = F(t_{j+1} - u_i) - F(t_j - u_i), \quad (2)$$

with $u_i = \beta^T x_i + \beta_0$, $t_{j+1} = t_j + \delta_j$ and $F(x) = (1+e^{-x})^{-1}$. We shifted the thresholds because our observations go from 0 to $m-1$, not from 1 to $m$.

Similar as before, we derive log likelihood as:

$$l = \sum_i \log(p_i(y_i)) =$$

$$= \sum_i \log((1+e^{-(t_{y_i+1} - \beta^T x_i - \beta_0)})^{-1} - (1+e^{-(t_{y_i} - \beta^T x_i - \beta_0)})^{-1}),$$

and input its negative value, together with initial guess ($\beta_i$ being 0.5 and $\delta_i$ being 1), training data and bounds for deltas.

We predict in the same way as in MLR, with probabilities from equation 2.

## Results

In this section, we will provide answers to the tasks above, and show some graphical results.

### Analysis of methods on main dataset

In the initial analysis we took first 80% of the main dataset for training, and last 20% for testing. We reached accuracy of 58% for multinomial, and 62% for ordinal.

Next we did cross validation on the whole dataset in which we were estimating the log loss of both implemented methods, compared with naive classifier, that always predicts probabilities $p = (0.15, 0.1, 0.05, 0.4, 0.3)$. In table 1 we can see the results of 10-fold cross validation.

**Table 1.** Mean and standard deviation of log loss of different models.

| | Naive | Multinomial | Ordinal |
|---|---|---|---|
| Mean | 1.34 | 1.27 | 1.19 |
| Std | 0.12 | 0.17 | 0.13 |

To better understand the method, we then analyzed the coefficients of built OLR model. We performed bootstrap - randomly sampled (with replacement) observations from the dataset, and trained OLR model on the sampled dataset of the same size as original. We made 1000 repetitions and saved values of $\beta$, then calculated the mean and 95% confidence interval for each coefficient. We can see the results on Figure 1.

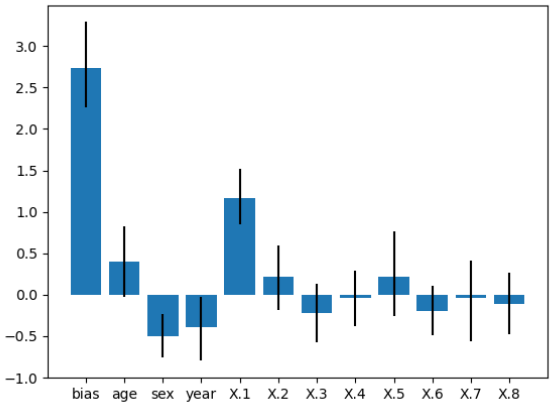


**Figure 1. Analysis of coefficients in OLR.** Mean value of $\beta$ coefficients corresponding to elements from input data. Blue columns represent mean value, and black line represencts 95% confidence interval. From it we can see, that without bootstraping we could easily make a mistake and interpret the coefficients the wrong way - a slight change in the training data can make a big difference in their values.

The first column, bias (/intercept), tells us what is the value of $u_i$, when all the input data equals 0. Since we standardized the data, this is exactly the value of $u_i$ for an average student. From other columns, we can see that older students generally rate the course better, those who are in second year rate it worse, the same goes for females. When we look at exam grades, we can see that the grade of the first exam is the most important of our input parameters. But for the others, we can't really say how are they connected to our prediction, since they are very unstable. To get an explanation why that happens, let's look at Figure 2.

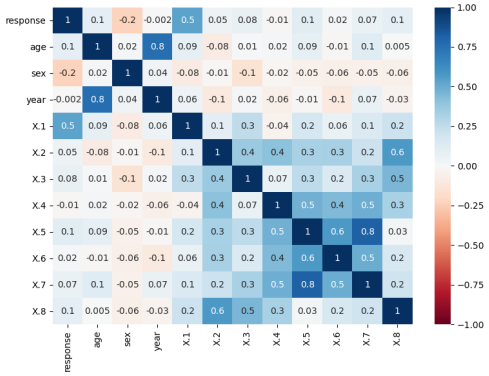


**Figure 2. Correlation in the data.** We can observe that X.1 has the biggest correlation with response, which corresponds to figure 1. There is also very big correlation between age and year, which is also logical. That also explains big CI at those two parameters. The part that we are most interested in, is the correlation between scores on the exams. We can see that their correlation is high, specially from X.2 on, which explains the big CIs. When the correlation between our input parameters is high, the model is not uniquely built, since we can have many options of $\beta$ that bring us to the same performance.

### Creating new dataset

MLR has more parameters to adjust - in our case 48, compared to 15 in OLR. That means it is more complex, so when we have enough data it usually performs better, but if our training dataset is small, it is not possible to correctly set this many parameters. This is why, for the dataset on which we want MLR to be bad, and OLR to be better, we generated only 10 datapoints for training data, and (as stated) 1000 points for testing.

We generated it by randomly sampling $y_i \in \{0,1,2,3,4\}$ and then calculating $x_1(y) = 2^y, x_2(y) = 3y^3 + (y-3)^3, x_3(y) = -y + (y-1)^2 - y^3 + e^y, x_4(y) = e^{y/5 - y^2 + (y-3)^{3/23}}$. We do that on each point and add some uniform noise between -1 and 1 to the values. An important thing to check when randomly generating such small dataset was also that our training data contained all classes.

On the train and test datasets generated like this, MLR reaches accuracy of 35.5% and mean log loss of 25.29, ORL has accuracy 46% and mean log loss of 1.63 and to compare it with naive classifier: accuracy of 22% and log loss 1.84.