HW3 Report
Logistic regression

Subject :
CS 6242 Spring 2017 - OMS

Author :
Melisande Zonta Roudes
GT account name : mzr3

March 30, 2017

1 Data Preprocessing

The dataset provides 785 rows (784 rows plus one for labels) and 24217 columns for train (4017 for test). This dataset is inverted indeed usually the attributes stand in the columns and the training examples in the rows. Our data are images so the 784 attributes are pixel (image of 28×28). Some operations have to be done to give the appropriate representation of these images. The images are handwritten figures : 0, 1, 3 and 5. It will be the basis of a classification survey between those figures, a binary one.

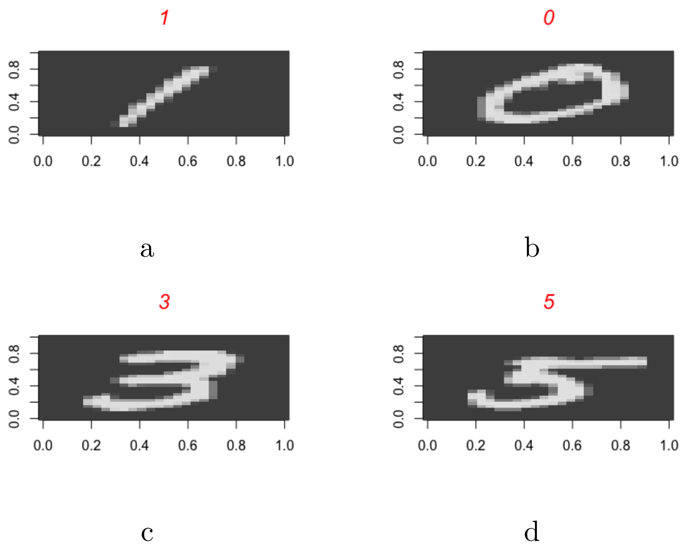


Figure 1: *a.* Image digit 1. *b.* Image digit 0. *c.* Image digit 3. *d.* Image digit 5

2 Theory

a. Demonstration The logistic regression which will be covered here is the one using the batch gradient descent method.

In this part, we will use several notations :

- n the number of examples
- d the dimensionality
- x the vector of input $\mathbf{x} \in \mathbb{R}^{n \times d}$
- y the vector of labels which in our case 0/1 $\mathbf{y} \in \mathbb{R}^{n \times 1}$
- θ vector of weights $\theta \in \mathbb{R}^{d \times 1}$

We will consider the logistic function known as sigmoid function

$$g(z) = \frac{1}{1 + e^{-z}}$$

The shape of this function shows that it is bounded between 0 and 1. In the hypothesis $h_\theta(x)$, we know that

$$h_\theta(x) = g(\theta^T x)$$

One of the main interest of the sigmoid function is that it is derivable :

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}(e^{-z}) \\ &= \frac{1}{1+e^{-z}} \times (1 - \frac{1}{1+e^{-z}}) \\ &= g(z)(1-g(z)) \end{aligned}$$

In order to perform true logistic regression on our data we put in place the convention of x_0 equal to 1 so that

$$\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

In order to fit our θ to the logistic regression model, let's write the probabilistic assumptions

$$\begin{aligned} P(y=1|x;\theta) &= h_\theta(x) \\ P(y=0|x;\theta) &= 1 - h_\theta(x) \end{aligned}$$

which can be written with a Bernouilli formula as

$$p(y|x;\theta) = (h_\theta(x))^y (1-(h_\theta(x)))^{1-y}$$

. In order to determine the maximum likelihood, we determine first the likelihood of the parameters. Indeed as the training examples are independent, we obtain :

$$\begin{aligned} L(\theta) &= p(y|X,\theta) \\ &= \prod_{i=1}^{n} p(y_i|X_i,\theta) \\ &= \prod_{i=1}^{n} (h_\theta(x^i))^{y^i}(1-h_\theta(x^i))^{1-y_i} \end{aligned}$$

In order to determine the maximum, we transform the previous expression into a logarithmic one.

$$\begin{aligned} l(\theta) &= \log(L(\theta)) \\ &= \sum_{i=1}^{n} [y_i \log(h_\theta(x^i)) + (1-y_i)\log(1-h_\theta(x^i)))] \end{aligned}$$

By normalizing the function, we obtain the loss function. Since we want to perform gradient descent, we are trying to minimize the negative likelihood.

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(h_\theta(x^i)) + (1-y_i)\log(1-h_\theta(x^i))]$$

Now we have the expression of the negative log likelihood, we differentiate the expression so we can find the maximum.

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta_j} &= -\frac{1}{n}[(y_i\frac{1}{h_\theta(x^i)} - (1-y_i)\frac{1}{1-h_\theta(x^i)})\frac{\partial h_\theta(x^i)}{\partial \theta_j}] \\
&= -\frac{1}{n}[(y_i\frac{1}{h_\theta(x^i)} - (1-y_i)\frac{1}{1-h_\theta(x^i)})h_\theta(x^i)(1-h_\theta(x^i))\frac{\partial \theta^T x^i}{\partial \theta_j}] \\
&= -\frac{1}{n}[(y_i(1-h_\theta(x^i)) - (1-y_i)h_\theta(x^i))x_j^i] \\
&= -\frac{1}{n}[(y_i - h_\theta(x^i))x_j^i]
\end{aligned}$$

Therefore the θ update is :

$$\theta = \theta - \alpha \nabla_\theta J(\theta)$$

with the gradient of the loss function written as :

$$\nabla_\theta J(\theta) = -\frac{1}{n}[(y - h(\theta^T x))x^T]$$

b. Pseudo Code The implementation of the logistic regression includes several steps like the calculation of the cost function which is the derivative of the loss function and then the update of our weights vector.

Algorithme 1 Training a model using Logistic Regression.

```
1: Binarize the labels of the dataset (conversion to 0/1)
2: Add a column of ones to X which represents the constant X = [ones(n,1) X]
3: Initialisation of the weights vector $\theta = [0, 0, ..., 0]^T$
4: h(x) = $\frac{1}{1+e^{-x}}$
5: while convergence-criterion > threshold do
6:     $\theta = \theta - \alpha \times \nabla_\theta J(\theta)$
7:     $\alpha = \alpha \times 0.9$
```

As described in the videos, a decreasing learning rate was chosen in order to accelerate the convergence and avoid a possible overfitting.
The stopping criteria of this algorithm are various but two of them caught my attention.

- The most natural one seems to be to compute the norm 2 (euclidean norm) of the difference between two successive vector θ. In this way, we come up with the computation of the euclidean norm of the multiplication of the learning rate and the cost function. We stop when a threshold is reached on this criteria or when the maximum of iterations is reached (in the case of a non convergence).

- The second one is linked to the first one since we will consider the difference of values of the logistic loss over the successive iterations. Indeed, it's interesting to consider the loss function which is the primitive of the cost function which gives different informations than the cost function multiplied by α.

c. Other details With d the dimensionality of θ and n the number of training examples, we consider the gradient descent function which calls cost-function and sigmoid functions : 4 operations are computed in the sigmoid function, $(y - h(\theta^T x)$ in the gradient is computed in n(8d-3) operations since the sigmoid function is applied to the 2d-1 operations for $\theta^T X_i$ (so 8d-4) plus we do 1 operation for the substraction and finally it is done for each training example, $(y - h(\theta^T x))X^T$ is adding d(2n-1)operations, then the normalization by n adds 3d operations and finally the decrease of the learning rate adds 2d operations. Hence the complexity is $O(n \times d)$.

3 Implementation

In order to implement the pseudo code below, three functions were created :

1. Sigmoid function
2. Cost Function
3. Gradient descent with the two types of convergence criteria mentioned above

4 Training

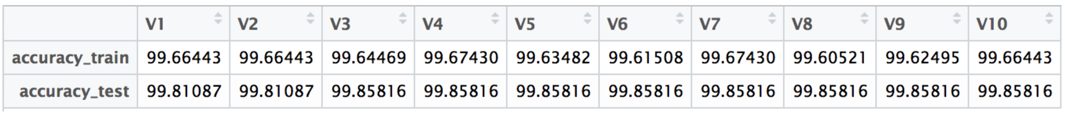
a. The convergence criterion chosen is the first one (euclidean norm of the difference of θ) and the threshold is set to 10^{-3} and the maximum iterations is 3000. In order to evaluate the algorithm we have implemented before, a function accuracy was created. It's role is to compare the output created from the weights vector θ and to compare it with the dataset's labels.

Table 1: Training and testing accuracies over one iteration

	Training Accuracy	Test Accuracy	Number of iterations
Classification 0/1	99.65259 %	99.85816 %	30
Classification 3/5	92.62465 %	94.00631 %	41

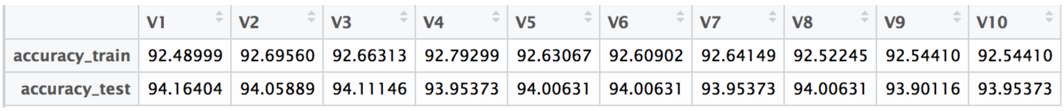
As we can see in the table 1, the accuracies are really high for the dataset 0/1 with an higher value for the testing than the training. The classification 3/5 is not performed as well as 0/1, but the increase in the testing rate is much more visible. We can also see the difficulty to converge since the number of iterations is higher in the classification 3/5 than in the 0/1.

b. As highlighted in the notes of this question, I encountered the classical problem of the batch method which is that over 10 iterations, we will obtain the same accuracy. To solve this problem, we randomly shuffle the data by sampling on 80%.



	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
accuracy_train	99.66443	99.66443	99.64469	99.67430	99.63482	99.61508	99.67430	99.60521	99.62495	99.66443
accuracy_test	99.81087	99.81087	99.85816	99.85816	99.85816	99.85816	99.85816	99.85816	99.85816	99.85816

a.



	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
accuracy_train	92.48999	92.69560	92.66313	92.79299	92.63067	92.60902	92.64149	92.52245	92.54410	92.54410
accuracy_test	94.16404	94.05889	94.11146	93.95373	94.00631	94.00631	93.95373	94.00631	93.90116	93.95373

b.

Figure 2: *a.* Classification between 0 and 1 over 10 iterations. *b.* Classification between 3 and 5 over 10 iterations.

We can visualize on the figures 2 that the accuracies are different so the technique is working, we don't obtain a value repeated 10 times.s

Table 2: Training and testing accuracies over 10 iterations

	Training Accuracy	Test Accuracy
Classification 0/1	99.64666 %	99.8487 %
Classification 3/5	92.61335 %	94.01157 %

The results are really close as we can see but as their represent an average over multiple iterations, they are more trusted values than random results that occur on one iteration.

c. The difference between the classification of 0/1 and 3/5 is directly linked to the shape of our figures we have seen in the first part. Indeed, the figures 0 and 1 have nothing in common, so the number of pixels in common is really small whereas the curve present in the low part of 3 and 5 are absolutely similar, which makes of those figures half similar. From those observations, we can understand the difference of results in accuracy and the difficulty to converge towards our threshold.

d. Although this logistic regression task was designed for binary classification, several classes can be handled also by this method. A common strategy is to use multiple binary classifiers to decide on a single-best class for new instances. We may create a classifier for each class in a one-versus-all way then, for new points, classify them based on the classifier function that produces the largest value. An other method could be to set up classifiers comparisons and select class that 'wins' the most pairwise matchups for new points.

5 Evaluation

In order to perform well on our classification, some parameters can be tuned as the initialization vector and all about the stopping criteria (maximum of iterations, threshold or convergence criteria). In this section, we will do some tests on the parameters but the maximum of iterations and threshold remain the same (max.iterations = 3000 and conv.criterion = 10^{-3}) in order to make some suitable comparisons with our previous results.

a. The initial weight vector was a vector of zeros. As the input values are bound between 0 and 1, we will determine if it's easier to converge if $\theta = [0, 0, ..., 0]^T$ or with a θ composed randomly of 0's and 1's.

Table 3: Change in the initialization of the weight vectors

	Training Accuracy	Test Accuracy	Number of Iterations
1 iteration	83.85561 %	85.75184 %	51
10 iterations	85.19965	86.49317	

As a recall, the number of iterations for 3/5 classification was 41 and the accuracy was around 92-94 % so this initialization vector represents a huge obstacle to converge and the accuracy obtained (around $83 - 86\%$) is worse than with the baseline criterion.

b. As detailed in the pseudocode, the first convergence criterion was based on the cost function whereas the second is on the loss function. It evaluates different behaviours.

Table 4: Change in the convergence criteria

	Training Accuracy	Test Accuracy	Number of Iterations
1 iteration	92.19183 %	93.74343 %	17
10 iterations	92.253 %	93.75394 %	

As we can see on the table 4, the accuracies are rather the same than the baseline results. There is a slight difference indeed the second criterion provide less good results that the first one but the number of iterations is remarkably lower (17 on the second instead of 41 for the first one).

6 Learning Curves

In conclusion of these tests, the best way to experiment the behaviour of our logistic regression algorithm on our dataset is to draw learning curves (training curves and testing ones). In this part, we will test our accuracy and logistic loss on different splits of our dataset. The splits were done 5 by 5 from 5% to 100%.

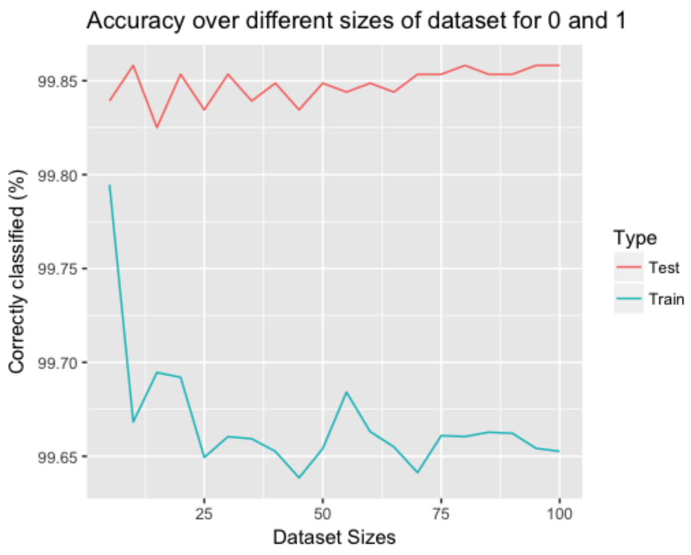


Figure 3: Learning curve for the classification between 0's and 1's

a. Accuracy curves The training curves and testing curves on figure 3 do not provide much information since we can see that the test accuracy is constant (around 99.85% for the testing accuracy and between 99.65% and 99.80% for the training) over the different dataset sizes and although the train set seems to have a different variation, the peaks are not relevant. By looking at the range of values, we can see that the minimal and maxima are really close. The second striking observation is the fact that the test accuracy is much higher than the train one which is unusual. This could be explained by the fact that in our code we randomly shuffle split our train data in order to obtain our θ from the gradient descent method whereas we keep the test dataset the same, no shuffle split is done. So from these observations we could say that the test set is easier for the classification between 0 and 1 in comparison to the shuffled split training dataset.

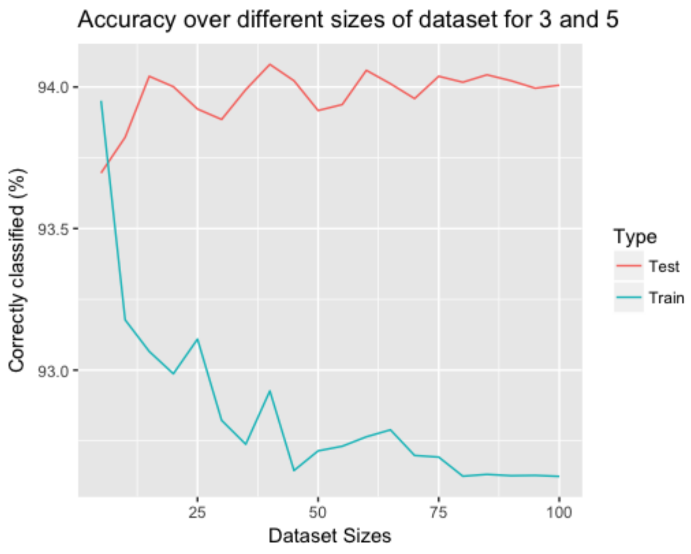


Figure 4: Learning curve for the classification between 3's and 5's

The training and testing curves on figure 4 for the 3/5 classification are a lot more interesting. Just by observing the range we can see that this one is more relevant (around 1.5% of variation) than the 0/1 classification indeed for the testing the accuracy goes from 93.54% to 94.0% and for the training it went down from 94.0% to almost 92.5%. We notice a decreasing behaviour of the train set and a slightly increasing variation of the test set. This can be explained by the fact that the more data the learner has to train one the less the accuracy is good. Indeed, training on 1210 examples is easier than training on 24217 examples. The test accuracy has a logical behaviour since the more the learner trains, the better the test is successful. The inversion of train and test curves has the same explanation mentioned above.

b. Logistic loss curves As the accuracies were meaningful, the logistic loss is another representative parameter. Indeed loss functions represent the price paid for inaccuracy of predictions in classification problems.

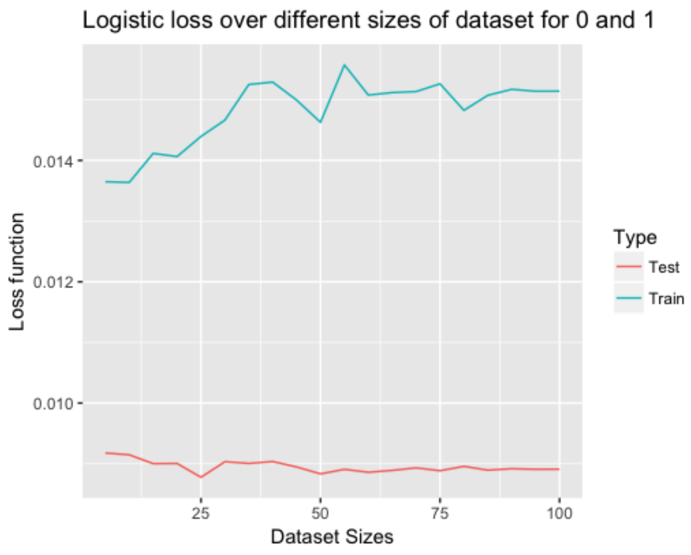


Figure 5: Logistic loss curve for the classification between 0's and 1's

The constancy of the training and testing curves on figure 5 (for the testing around 0.009 and between 0.014 and 0.015) has been mentioned in the first part on the accuracy training and testing curves. The difference between the previous curves and these ones stand in the inversion between the training set curve and the testing set curve, which is logical since the loss function represents somehow the inaccuracy.

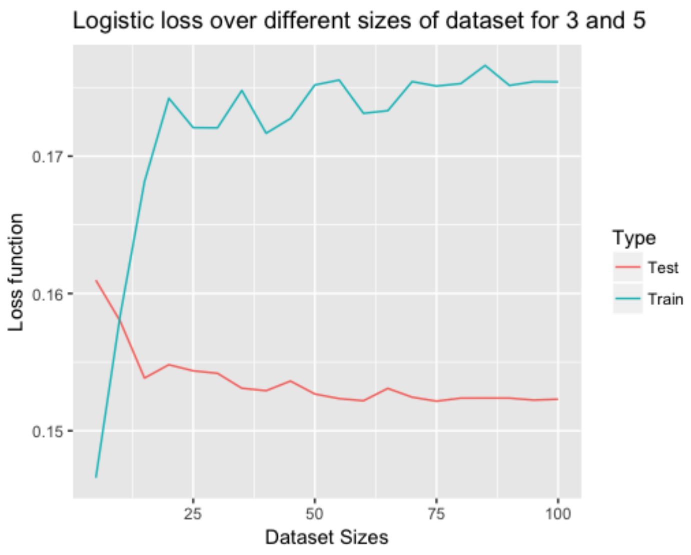


Figure 6: Logistic loss curve for the classification between 3's and 5's

The inversion of the training curves and testing curves can also be seen on figure 6, and we can observe as well the increasing behaviour of the train set and the decreasing one for test set. The testing accuracy goes from 0.16 to 0.152 and the training accuracy from 0.148 to 0.175.