TDT4173 - Assignment 1
Logistic regression and K-means algorithms

Arthur Testard - id: 105022

1 Logistic Regression

1.1 What's this algorithm

This algorithm is a very simple model which allows to classify data. The target variable $y \in \{0, 1\}$ is known so we talk about a supervised learning algorithm. The input is $x \in R^n$ where $n \in N$ is the number of features. We call h_w our logistic regression model, where $w \in R^n$, which is defined such as following:

$$\hat{y} = h_w(x) = \sigma(w^T x) = \frac{1}{1 + e^{w^T x}}$$

Where, $\hat{y}$ is the prediction of y, and σ is the logistic function which is the sigmoid function in this model.

Our goal is to find the good parameter W which fit our dataset $X = \{x^{(j)}\}_{j \in \{1,...,m\}} \in R^{m \times n}$, with m the number of features. So we approximate our model like this, written for each $j \in \{1, ..., m\}$, given that $x_0 = 1$ and $w_0 = b \in R$:

$$h_w(x^{(j)}) = \sum_{i=0}^{n} w_i x_i^{(j)} = \sum_{i=1}^{n} w_i x_i^{(j)} + b = w^T x^{(j)} + b$$

We then define the cost function that we want to minimize, applied on all of the samples:

$$J(w) = \frac{1}{2} ||h_w(X) - y)||$$

The chosen norm here is the euclidean one. Then our gradient descent can be written this way, after calculus, with α is the learning rate:

$$w_i := w_i + \alpha \sum_{j=1}^{m} (y^{(j)} - h_w(x^{(j)})) x_i^{(j)}$$

$$b := b + \alpha \sum_{j=1}^{m} y^{(j)} - h_w(x^{(j)})$$

For the following, we introduce some orders on our logistic regression's model by adding input on it. To do so, we change our input x (one sample) by a new one which is on order $d \in N$ can be written this way:

$$x = (x_1, x_1^2, ..., x_1^d, ..., x_n, ..., x_n^d)^T$$

1.2 Inductive bias

An inductive bias in the primary model, without the d-order, is that we consider the data is generated via a linear process. We will see that it works for the first example. However, it will not for the second.

1.3 Some plots

The whole code to plot this figures can be found in the code files.

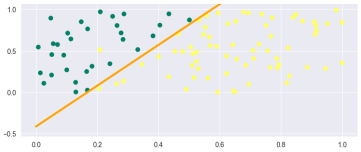

Figure 1: LR - Model 1 plot

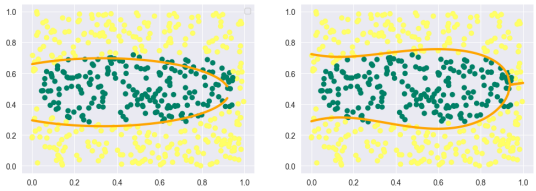

Figure 2: LR - Model 2 plots - order 2 (left) and 3 (right)

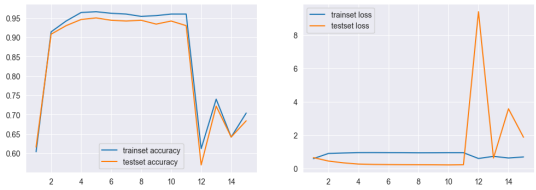

Figure 3: LR - Model 2 accuracies and losses plot by orders

1.4 Results/preprocessing

Concerning the first dataset, the model is converging quite quickly: 50 epochs to get 90% of accuracy with a learning rate of 0.1. We get Figure 1 with initial weights get manually, with random one the result is a bit worth (93% of accuracy at the end, see code).

Concerning the second dataset, we had to increased the order of the input because the data were not generated via a linear process. We get then on Figure 2 how does this models looks (we could plot in 3d to have a better visualisation). We can see on Figure 3 that we have an accuracy $> 90\%$ on both train and test sets for orders between 2 and 11 but then it decreases. At the same time, the loss of the test set which increase a lot but not the train set's one. Two things: the model is over-fitting or it requires more epochs to fit the dataset.

2 K-means

2.1 What's this algorithm

In this problem, unlike the algorithm of Logistic regression, we do not have any output y in the dataset. So you are talking about an unsupervised learning algorithm. However, we have an input $X = \{x^{(j)}\}_{j\in\{1,\dots,n\}} \in R^{n\times d}$, where $n \in N$ the number of samples and $\in N$ the number of features. The goal consist in finding k clusters -groups of samples close- in the dataset. We define the function of distortion measure:

$$J = \sum_{i=1}^{n}\sum_{j=1}^{k} r_{ij}||x^{(i)} - \mu^{(j)}||^2$$

Where, for each $(i,j) \in N^{n\times k}$, $r_{ij} \in \{0,1\}$ and $r_{ij} = 1$ if and only if $x^{(i)} \in j$-cluster. Thus, our goal is to minimize this function. Therefor, we can assume that $x^{(i)} \in j_0$-cluster if and only if $r_{ij_0} = \arg\min_j ||x^{(i)} - \mu^{(j)}||$.

The basic algorithm is describes as follows (it can be improve in many ways as we will see):

1. initialize $\mu^{(j)}$ with random values
2. associate every $x^{(i)}$ to one cluster each, (ie. compute every r_{ij})
3. relocate every $\mu^{(j)}$ such as:

$$\mu^{(j)} = \frac{\sum_{i=1}^{n} r_{ij}x^{(i)}}{\sum_{i=1}^{n} r_{ij}}$$

4. loop on step 2 and 3

We then improved this algorithm, such as described by [BN06], by making different tries of the algorithm with different $\mu^{(j)}$ initialized at the beginning and taking the final set of $\mu^{(j)}$ which minimize the most our distortion function. This improve does not concern the first dataset as we will see, but it will be important for the second one.

2.2 Inductive bias

The inductive bias here is that cases that are close to each other tend to belong to the same class.

2.3 Some plots

The whole code to plot this figures can be found in the code files.

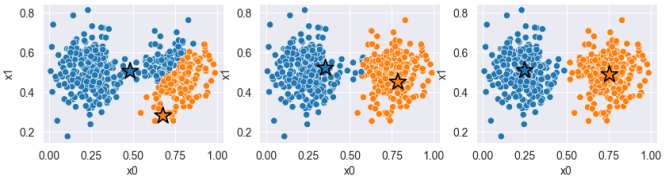

Figure 4: KM - Model 1 plots from epoch 1 to 3

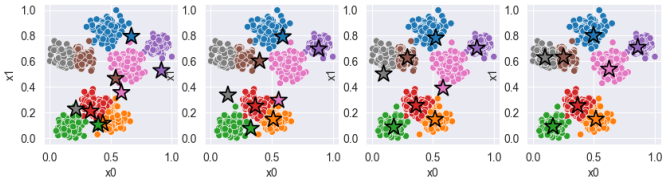

Figure 5: KM - Model 2 plots from epoch 1 to 3

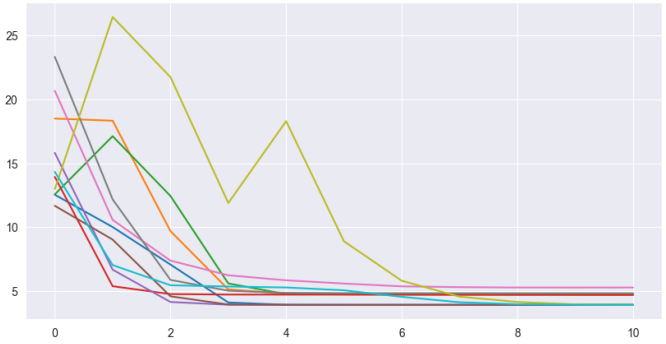

Figure 6: KM - Model 2 convergence with different initial $\mu^{(j)}$ ($tries = 10$ and $max\_epochs = 10$)

2.4 Results/preprocessing

With the first dataset we did not use the improve mentioned in the algorithm description because it was not needed. We can see that, with many tries and various different initial $\mu^{(j)}$, it always converge on the same solution very quickly. On Figure 4 that the algorithm converge on the solution on 3 steps.

Concerning the second dataset we had to use the improve mentioned before. The initial $\mu^{(j)}$ has a big impact on the primary algorithm and makes the algorithm converge on local minimum of the distortion measure function which is not satisfying. Thus, making many tries on it and taking the cluster which minimize the distortion function works pretty well. With our tries, we found out that making 20 tries is enough to find at least one time the $\mu^{(j)}$ which makes $J \approx 3,917$ (which seems to be the global minimum applied to this dataset).

]Figure 6 shows us how the model converge with different and randomized initial states. So we can assume that this implementation can finds the 8 clusters of the dataset at least 9/10 times with randomized initialization. Although, we could have make a condition on a while loop like to continue until finding a solution where $J \approx 3,917$ but we wanted to implement the most generalized algorithm as possible (the algorithm would then be too adapted to the problem at hand and not general enough).

Results of optional steps can only be found in the notebook.

References

[BN06] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.