
ML3 - Practical Class n°1

This practical class contains a reminder on logistic regression and basic gradient descent techniques.

Exercise n°1 : Logistic regression by gradient descent

In this exercise, we will use logistic regression to solve a binary classification task. The parameters of the model will be learned by batch gradient descent and its stochastic version.

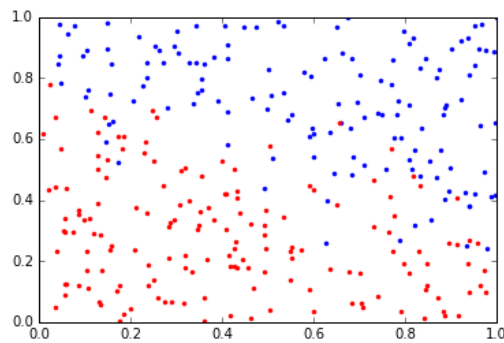
Questions :

1. Create a `datagen` function allowing to create a dataset with inter-class overlapping.

To that end, draw inputs uniformly in the unit square. This implies that inputs $\mathbf{x} \in \mathbb{R}^2$ and $0 \leq x_1, x_2 \leq 1$. To create an overlap, the class label of each generated input must be chosen as follows :

- determine the signed distance¹ D between the generated input and the line whose equation is $x_2 = -0.5x_1 + 0.75$,
- if $D > 0$, assign class label 1, otherwise assign class label 0,
- compute $r = e^{-\frac{D^2}{2\sigma^2}}$ with $\sigma = 0.05$,
- draw a sample $Z \sim \text{Ber}\left(\frac{r}{2}\right)$,
- if $z = 1$ then, change the class label.

Calling this function to generate a dataset with $n = 300$ points should give something like this :



2. Create a python variable `X_plus` by concatenating `X` with a line of 1 as :

$$\mathbf{X}_+ = \begin{pmatrix} \begin{bmatrix} \mathbf{x}^{(1)} \\ 1 \end{bmatrix} & \cdots & \begin{bmatrix} \mathbf{x}^{(n)} \\ 1 \end{bmatrix} \end{pmatrix}. \quad (1)$$

3. Let us start training a logistic regression by gradient descent. Logistic regression is meant to find a separating hyperplan (a line in our case). This hyperplan in question is parametrized by a normal vector \mathbf{w} and a constant b , called *intercept*. To have more compact notations, let us introduce the following

1. See section 4.1.1 p. 181 of [Bishop].

vector of parameters

$$\boldsymbol{\theta} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \\ b \end{bmatrix}. \quad (2)$$

The classical gradient descent, by opposition to the stochastic one, is called *batch* descent meaning that all data is used to compute one gradient vector. Subtracting this gradient to the vector of parameters allows to decrease the NLL / train loss.

```
Initialize  $\boldsymbol{\theta}_0$  and  $\eta$ .
while not converged do
    Compute predictions :  $\mathbf{pred} \leftarrow \text{sgm}(\mathbf{X}_+^T \cdot \boldsymbol{\theta}_t)$ .
    Compute signed errors :  $\mathbf{err} \leftarrow \mathbf{pred} - \mathbf{y}$ .
    Compute gradient :  $\mathbf{g} \leftarrow \mathbf{X}_+ \cdot \mathbf{err}$ 
    Update parameters :  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \times \mathbf{g}$ .
end while
```

For the time being, the *learning rate* η has a constant value.

Implement this procedure in python so that it returns the whole history of parameter vectors $\boldsymbol{\theta}$ across iterations. This history is stored in a 2D numpy array called `thetas` whose size is $3 \times$ the number of iterations.

4. Try your implementation with $\eta = 0.02$ and $\eta = 0.1$. Visualize the results thanks to the following code :

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot(thetas[0], thetas[1], thetas[2], '-o')
plt.draw()
```

5. Let us go stochastic. The procedure becomes :

```
Initialize  $\boldsymbol{\theta}_0$  et  $\eta$ .
while not converged do
    Draw a permutation  $\sigma$  at random.
    Use this permutation on examples :  $\mathbf{X}_+ \leftarrow \sigma(\mathbf{X}_+)$ 
    Use the same permutation on class labels :  $\mathbf{y} \leftarrow \sigma(\mathbf{y})$ 
    for  $i$  from 1 to  $n$  do
        Compute prediction for the  $i^{\text{th}}$  example :  $\text{pred} \leftarrow \text{sgm}(\mathbf{x}_+^{(i)T} \cdot \boldsymbol{\theta}_t)$ .
        Compute its signed error :  $\text{err} \leftarrow \text{pred} - y^{(i)}$ .
        Compute its gradient :  $\mathbf{g} \leftarrow \mathbf{x}_+^{(i)} \times \text{err}$ 
        Update parameters :  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \times \mathbf{g}$ .
    end for
end while
```

Implement this procedure in python and store the history of values of $\boldsymbol{\theta}$ and the number of « epochs² ». Indication : For the permutation, you may use `numpy.random.permutation`.

6. Try your implementation with $\eta = 1.5$ Visualize the trajectory of $\boldsymbol{\theta}$ with the same 3D plot as before.
7. Compare the results by displaying the learned separating lines on the scatter-plot of the data. Also compare the number of « epochs » in SGD to the number of iterations of the batch method.
8. Propose a modification (few lines of code) to the SGD algorithm so that gradients are computed from a mini-batch of 10 examples.

Proof of gradient formula for log. reg.

Recall from the class that the logistic regression loss for example $\mathbf{x}^{(i)}$ is given by

$$\ell_i = -y^{(i)} \log \text{sgm}(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)}) - (1 - y^{(i)}) \log(1 - \text{sgm}(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)})).$$

2. The number of times each data point was used, i.e. the number of times the for loop is executed.

Let us differentiate ℓ_i w.r.t. the vector $\boldsymbol{\theta}$:

$$\frac{d\ell_i}{d\boldsymbol{\theta}} = -y^{(i)} \frac{d}{d\boldsymbol{\theta}} \log \text{sgm} \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right) - (1 - y^{(i)}) \frac{d}{d\boldsymbol{\theta}} \log \left(1 - \text{sgm} \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right) \right), \quad (3)$$

$$= -y^{(i)} \frac{d}{d\boldsymbol{\theta}} \text{sgm} \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right) \frac{1}{\text{sgm} \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right)} - (1 - y^{(i)}) \frac{d}{d\boldsymbol{\theta}} \left(1 - \text{sgm} \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right) \right) \frac{1}{1 - \text{sgm} \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right)}. \quad (4)$$

Given that $\text{sgm}' = \text{sgm}(1 - \text{sgm})$, we have

$$\frac{d\ell_i}{d\boldsymbol{\theta}} = -y^{(i)} \underbrace{\frac{d}{d\boldsymbol{\theta}} \left\{ \boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right\}}_{=\mathbf{x}_+^{(i)}} \frac{\text{sgm}' \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right)}{\text{sgm} \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right)} - (1 - y^{(i)}) \frac{d}{d\boldsymbol{\theta}} \left\{ \boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right\} \frac{-\text{sgm}' \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right)}{1 - \text{sgm} \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right)}, \quad (5)$$

$$= \mathbf{x}_+^{(i)} \left[-y^{(i)} \left(1 - \text{sgm} \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right) \right) + (1 - y^{(i)}) \text{sgm} \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right) \right], \quad (6)$$

$$= \mathbf{x}_+^{(i)} \left[\underbrace{\text{sgm} \left(\boldsymbol{\theta}^T \cdot \mathbf{x}_+^{(i)} \right) - y^{(i)}}_{\text{pred}} \right]. \quad (7)$$

Finally,

$$\frac{d\text{NLL}}{d\boldsymbol{\theta}} = \sum_i \frac{d\ell_i}{d\boldsymbol{\theta}}, \quad (8)$$

$$= \mathbf{X}_+ \cdot \mathbf{err}. \quad (9)$$