Exercise 2.0 — Logistic Regression and Probabilistic Graphical Models

Artificial Intelligence For Robotics

Tonci Novkovic and Sebastian Verling

Spring Semester 2017

1 Logistic Regression

In this exercise you will write an algorithm to detect if the eye shown in a 24×24 pixel image is open or not by using logistic regression. An example of these images can be seen in Figure 1a and 1b

(a) Open eyes (b) Closed eyes

Figure 1: Example pictures

Remember that the logistic function $h(\boldsymbol{x}, \boldsymbol{w})$ and its cost function J for logistic regression is given as:

$$h(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}}} \tag{1}$$

$$J(w) = \sum_{i=1}^{i=n} -y^{(i)} \ln(h(\boldsymbol{x}^{(i)}, \boldsymbol{w})) - (1 - y^{(i)}) \ln(1 - h(\boldsymbol{x}^{(i)}, \boldsymbol{w})) \tag{2}$$

with $y^{(i)}$ and $\boldsymbol{x}^{(i)}$ being the reference and the feature vector of the i'th element and $\boldsymbol{w}$ being the feature weights. For the sake of simplicity we define the feature vector to be directly the pixel intensities. Remember that there is no analytic solution to the logistic regression for $\boldsymbol{w}$ and that we can do it iteratively using Newton's Method which is defined as:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{w}} J \tag{3}$$

with $\nabla_{\boldsymbol{w}} J$ being the first derivative of J with respect to $\boldsymbol{w}$ and $\boldsymbol{H}$ being the second derivative. Your goal is to implement an algorithm that iteratively computes the weight vector $\boldsymbol{w}$ to train your logistic regression for the open/closed eye classification.

Hint: The derivative of $h(\boldsymbol{x}, \boldsymbol{w})$ with respect to the j'th element of the feature vector w_j is

$$\frac{\partial h(\boldsymbol{x}, \boldsymbol{w})}{\partial w_j} = h(\boldsymbol{x}, \boldsymbol{w})(1 - h(\boldsymbol{x}, \boldsymbol{w}))x_j \tag{4}$$

Also, as we are dealing with real world data and we are using a rather simple algorithm, you can not expect the classifier to be perfect.

1.1 Tasks

Fill in the sections marked with #`TODO`.

1. `main.py`
    - implement the logistic function
    - initialize $\boldsymbol{w}$
    - implement the recursive calculation of $\boldsymbol{w}$

Once you have this implemented, you are very likely to see that the classification for the training data is very good, while the validation set classification is rather bad. This is likely to be due to over-fitting. Your next task is to adapt the cost function and therefore $\boldsymbol{H}$ and $\nabla_{\boldsymbol{w}} J$ of the Newton's Algorithm to account for regularisation.

2. adapt the cost function to account for regularisation
3. derive the new resulting $\boldsymbol{H}$ and $\nabla_{\boldsymbol{w}}$
4. `main.py` implement the recursive algorithm

2 Probabilistic Graphical Models

In this exercise you are going to implement a segmentation algorithm using a Probabilistic Graphical Model, specifically a Markov Random Field (MRF). Segmentation is widely used in computer vision and robotics in order to decompose the given image into smaller segments which either provide semantic meaning (i.e. objects, ground, people, etc.) or fewer units of more meaningful data that are easier to process afterwards.

The goal of the segmentation is to assign a meaningful label x to each pixel p in the image I. In general these labels could be semantic, e.g. person, car, tree, etc., or as in our case, they can be just numbers, e.g. 0, 1, 2, etc. Since the labels are initially unknown, they need to be determined based on the observations z (also called features), which in this case are nothing more than pixel intensity values for grayscale images or color intensities for color images (i.e. values we can observe). Additionally, we can define a probability of assigning a label x to the pixels in an image based on observation z as:

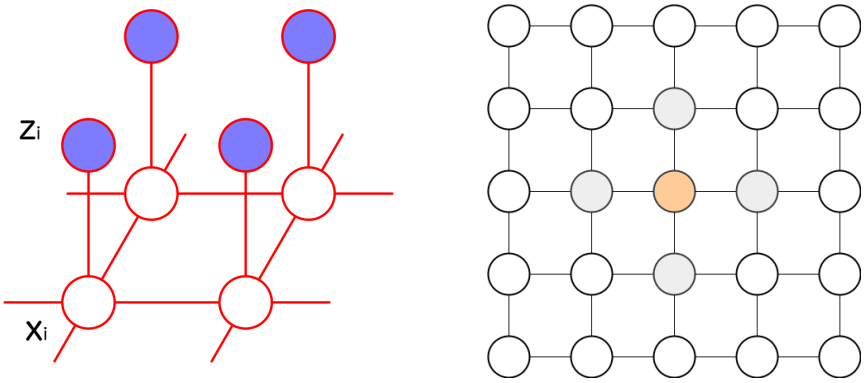
$$P(x|z) = \frac{P(z|x)P(x)}{P(z)} \propto P(z|x)P(x)$$

where $P(z|x)$ is likelihood of observing feature z given the label x, and $P(x)$ is the prior distribution of the labels. Now, we can define our goal more formally as finding an optimal labeling $\hat{x}$ which maximizes $P(x|z)$, i.e. finding Maximum a Posteriori (MAP) estimate:

$$\hat{x}^{MAP} = \arg\max_{x} P(x|z)$$

In order to solve this problem, we first need to define $P(x)$ and $P(z|x)$ through our model.

An image of MRF probabilistic graphical model is shown in Fig. 2a. The advantages of this model are twofold. First, it captures the contextual constrains often exhibited in images (e.g. homogeneous regions with similar color, intensity, etc.), and second, it is a generative model meaning that it can generate unseen observations which, as we will see later on, is a very useful property. In our case, since the probability of assigning any label x is strictly positive and as we can assume that probability of the label is only dependent on neighboring labels, we can model our problem as a MRF. This



(a) Markov Random Field model (b) Neighboring nodes in MRF

Figure 2: Markov Random Field graphical models represent joint probability distributions where, given its neighbors, a node is independent of all other nodes in a graph.

means that we can, based on Hammersley-Clifford Theorem[1], write the $P(x)$ as:

$$P(x) = \frac{1}{2}\exp(-U(x))$$

where $U(x)$ is the energy of the configuration x. This energy can also be expressed as a sum of all clique potentials (see lecture slides: cliques are fully connected subgraphs in the MRF). Since our MRF only contains 1st and 2nd order cliques (singletons and doubletons), total energy can be written as:

$$U(x) = \sum_{c \in C} V_c(x) = \sum_{i \in C_1} V_{C_1}(x_i) + \sum_{(i,j) \in C_2} V_{C_2}(x_i, x_j)$$

In order to obtain the clique potentials, we also need to define $P(z|x)$, which in our case can be modeled as a normal distribution:

$$P(z|x) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

If we assume that the number of labels in the image is known, $P(z|x)$ can be obtained using the Expectation Maximization (EM) algorithm. Clique

[1]Theorem that states the conditions for a random field to be a MRF, for more details see J.M.Hammersley, P. Clifford "Markov fields on finite graphs and lattices"

potential of the singletons is then defined as the log likelihood of the features given label:

$$V_{C_1}(x_i) = -\log(P(z|x_i))$$

and for the doubletons as:

$$V_{C_2}(x_i, x_j) = \beta\delta(x_i, x_j) = \begin{cases} -\beta & \text{when } x_i = x_j \\ +\beta & \text{when } x_i \neq x_j \end{cases}$$

where $j \in N$ and N is the neighborhood of i (see Fig. 2b) and beta is the parameter defining the strength of second order clique potentials. Finally, substituting the clique potentials, the energy function becomes:

$$U(x) = \sum_p (\log(\sqrt{(2\pi)^k \det \boldsymbol{\Sigma}}) + \frac{1}{2}(\boldsymbol{z_p} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{z_p} - \boldsymbol{\mu})) + \sum_{p,n} \beta\delta(x_p, x_n)$$

and therefore our goal becomes a minimization problem:

$$\hat{x}^{MAP} = \arg\max_x P(x|z) = \arg\min_x U(x)$$

Since we would like to find the global minimum and we are able to generate unseen samples, we can use simulated annealing, a stochastic optimization algorithm, to solve the problem. The algorithm initializes labels randomly and in each iteration constructs a perturbation (changing one label) of the current labels, computes the energy of the perturbed labels and decides if the new perturbed labels are kept for the next iteration or not. Furthermore, the temperature parameter T, which determines the acceptance probability of the perturbed labels, is decreased in each iteration meaning that it is less likely to accept new perturbations with higher energies. We can stop the iterations once the maximum number of iterations has been reached or once the change in energy between iterations becomes small enough.

2.1 Tasks

Fill in the sections marked with #`TODO`.

In `MarkovRandomField.py`

1. Implement `singleton` method

2. Implement `doubleton` method
3. Implement `calculateGlobalEnergy` method
4. Implement `calculateLocalEnergy` method

In `main.py` you don't need to write any code, you can just tune the parameters of the algorithm.

2.2 References

You can find more information on PGMs here:

1. Christopher M. Bishop: Pattern Recognition and Machine Learning
2. Slides by Zoltan Kato: Markov Random Fields in Image Segmentation `https://inf.u-szeged.hu/~ssip/2008/presentations2/Kato_ssip2008.pdf`

3 Submission

To hand in the exercise you have to zip the whole folder `(2_0_regression_pgm)` with your code and results files and upload it on moodle. The zip file you upload should have the name *aifr_familyname_ex2* (replace familyname with your family name). The python scripts are written in a way, such that it always creates a `results.txt` or `results.pkl` file with the data that is used for evaluation.