Assignment 2[1]. Learning Bayesian Networks and EM algorithm: An application to kinect data.

Objetive

In this assignment we are going to work with of **MSRC-12 Kinect gesture data set of Microsoft Research Cambridge**. This dataset consists in hundreds of sequences of skeletal body movements recorded from a kinect device. However, for this assignment only a small fraction of this dataset, consisting of static body positions, will be considered. Specifically, we will work in this assignment with the following body positions: crouch, right arm extended and both arms lifted.

The assignment has two options: classification and clustering. In the first option, two Bayesian models will be trained in order to classify the different body positions. The first model is Naive Bayes, that considers only dependencies from the class variable to the other random variables. In the second model, more elaborated, dependencies between the positions of the different parts of the body will also be considered.

In the second option of the assignment we will ignore the class labels of the instances and we will cluster the different positions using the EM algorithm considering the class variable as a hidden (unknown) variable.

Data structures and functions

The fraction of the dataset we will be working on consists in 2045 instances of body positions for 4 classes: “arms lifted”, “right arm extended to one side”, “crouched” and “right arm extended to the front”. The body positions are encoded using a 20x3 matrix where each row is the position in space (x,y,z) of each of the 20 joints. The order in which the different joints appear in the vector are shown in Figure 1 for one instance of class both arms lifted. The data is in a matlab file data.mat. To load the data in python use `scipy.io.loadmat` function. The content of the files is:

- data: a 3D matrix with dimensions #joints x 3 x #instances. In this case is 20x3x2045.
- labels: a vector with the class label for each instance.
- individuals: a vector with the identification of each individual.

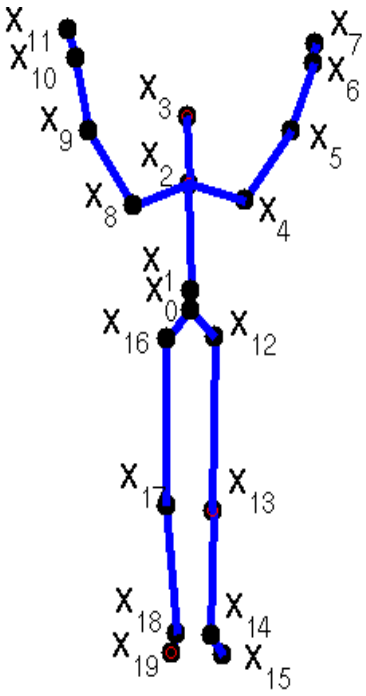

Figure 1

The following auxiliary functions are given. Note that, these functions are either from the MSRC-12 repository or adapted from those:

- **skel_vis**: (Not published yet) It plots the skeleton for a given 20x3 instance.
- **skel_model**: It loads the skeleton definition and relations between joints.

[1]This assignment is inspired on Daphne Koller Probabilistic Graphical Models course assignments.

Option A. Naive Bayes and Linear Gaussian Model

In the Naive Bayes model each of the 60 variables that define the body position are considered independent given the class. Each variable is going to be modeled using a Gaussian distribution. The training process for this model will consist in estimating by MLE the values for the mean and variance for each variable and class.

In the second model we will apply the Linear Gaussian Model using the structure of the body parts, so that:

$$p\left(x_i \mid x_{p(i)}, y_{p(i)}, z_{p(i)}, C\right) = Normal\left(\beta_{01} + \beta_{11} x_{p(i)} + \beta_{21} y_{p(i)} + \beta_{31} z_{p(i)}; \sigma^2\right)$$
$$p\left(y_i \mid x_{p(i)}, y_{p(i)}, z_{p(i)}, C\right) = Normal\left(\beta_{02} + \beta_{12} x_{p(i)} + \beta_{22} y_{p(i)} + \beta_{32} z_{p(i)}; \sigma^2\right)$$
$$p\left(z_i \mid x_{p(i)}, y_{p(i)}, z_{p(i)}, C\right) = Normal\left(\beta_{03} + \beta_{13} x_{p(i)} + \beta_{23} y_{p(i)} + \beta_{33} z_{p(i)}; \sigma^2\right)$$

where p(i) indicates the index of the joint that is the parent of the i-th joint.

The first task will be to implement two functions to estimate the parameters of the Gaussian distributions using maximum likelihood estimation:

- **fit_gaussian**: The input of this function is a vector of observations for a given variable and the output will be the mean and standard deviation for those observations. This is as simple as computing the mean and standard deviation of the sample.
- **fit_linear_gaussian**: This function has two input parameters. A vector Y with the observations for the variable of interest and a matrix X with the observations for the parent variables of Y. Both elements should have the same number of observations (i.e. rows). The output is a list with the estimated beta values and the standard deviation of the Gaussian. The beta values are obtained by solving a set of linear equations. This can be done in **python** with np.linalg.solve function (e.g. the linear equation Ax = b is solved by x = np.linalg.solve(A,b)).

The next task is to build a complete model for the dataset and not just a single variable estimation. For that we will implement:

- **learn_model**: The input of this function is the dataset and labels as obtained from load and, optionally, a skeleton structure, as loaded from skel_model. If no skeleton structure is given, the function will assume that all variables are independent given the class (naive Bayes model). Otherwise the linear Gaussian model is assumed. The structure of the output is defined in the .py file.
- **classify_instances**: The input of this function is a set of instances in the same format as the dataset given by load and the model as computed in learn_model. The output is the posterior probability for each instance belonging to each class:

$$P\left(C=k \mid instance\right) \propto P\left(C=k\right) \prod_{j=1}^{20} p\left(x_i, y_i, z_i \mid x_{p(i)}, y_{p(i)}, z_{p(i)}, C=k\right) \quad (1)$$

where

$$p\left(x_i,y_i,z_i|x_{p(i)},y_{p(i)},z_{p(i)},C=k\right)=p\left(x_i|x_{p(i)},y_{p(i)},z_{p(i)},C=k\right)\times$$
$$p\left(y_i|x_{p(i)},y_{p(i)},z_{p(i)},C=k\right)\times$$
$$p\left(z_i|x_{p(i)},y_{p(i)},z_{p(i)},C=k\right)$$

Important: to avoid underflow numerical issues this computations should be performed in log space. In this sense it is recommended to implement the following functions:

- **compute_logprobs**: The input of this function is one instance of the dataset and the model as computed in learn_model. This functions computes the log-probablity, that is, the logarithm of equation (1) for each class value k.
- **log_normpdf**: Computes the natural logarithm of the normal probability density
- **normalize_logprobs**: Receives a vector of log-probabilities and output a vector of the same size with values in [0,1] and that adds up to 1. This operation should be done to avoid underflow issues. Already implemented

Functions should be as generic as possible and take advantage of vector operations.

Option B. EM algorithm

The second option of the assignment, we have to implement a clustering of the poses into classes. For this option we are going to ignore the labels of the instances.

You have to implement the function:

- **em_pose_clustering**: The input of this function is the dataset as obtained from load_data, a matrix with of initial probabilities for each instance belonging to each of the classes in which we would like to cluster the instances and, optionally, a skeleton structure, as loaded from skel_model. If no skeleton structure is given, the function will assume that all variables are independent given the class (naive Bayes model). Otherwise the linear Gaussian model is assumed. The output of this function is the obtained model as defined in the .m file and the probability assignments of each instance.

This algorithm is divided into two main steps:

M-step: In this step the MLE of the model has to be computed using the soft assignments for the class. This step is similar to option A of this assignment but taking into account that each instance has a probability of belonging to a class label instead of a fixed class assignment. For this you will need to adapt fit_linear_gaussian to include one more parameter to handle the class weight of each instance. As a reference the fit_gaussian function is given. We will start with this step (instead of the E-step) in order to initialize the model using the probabilities given as parameter.

E-step: In this step a soft assignment of instances to classes has to be computed

using formula (1).

You might find useful implementing the following functions:

-log_normpdf: To compute the natural logarithm of the normal probability density function.

-log_sum_exp: To compute the log(sum(exp(X))) of the input X avoiding numerical issues when we must add probabilities that are currently in log space.

The results for this part are not necessarily very good. Using the complex model of 20 joints might not produce the best results for the clustering. You are encouraged to try different ideas to improve the clustering like simplifying the skeleton model, or eliminating variables, etc.

Submission:

Option A: the submission should include the code and a report. The report should include an brief explanation of the implemented methods, description of the evaluation procedure (eg using cross-validation), tables showing the classification performance of the different methods, description of the results and conclusions. Grading table as reference:

Code	
-Generality	1.5
-Tidyness	0.5
Code description & docum	1
Introduction	1
Description of evaluation	2
Results	1
Description of results	2
Conclusions	1

Option B: the submission should include the code and a report. The report should include an brief explanation of the implemented methods, description of the results and conclusions.

The deadline for this assignment is **April 12th**.