Probabilistic Graphical Models Programming Assignment #8: Learning Tree-structured Networks

This assignment is due at 11:59pm PDT (UDT -7) on 19 May 2012.

1 Introduction

In the past few weeks, you have learned about parameter estimation in probabilistic graphical models, as well as structure learning. In this programming assignment, you will explore structure learning in probabilistic graphical models from a synthetic dataset. In particular, we will provide you synthetic human and alien body pose data. We use a directed model to represent their body poses. You will tackle problems including learning CPDs for continuous variables and learning tree-structured graphs among body parts. Finally, you will use the learned models to classify unseen pose data as either humans or aliens.

2 Notation and Model Choices

2.1 Representation of Body Poses

We model the body parts with ten parts, as illustrated in Figure 1. The configuration of each body part is parameterized by 3 real numbers: (y, x, α), where (y, x) is the position of the anchor point of the body part (shown as filled black squares) and α is the orientation of the body part (shown as arrows). The y value increases from top to bottom; x increases from left to right; α increases clockwise and has the origin (zero value) pointing upright.

Given this setting, a pose of the full body can be specified by a 10×3 matrix, where the 10 rows correspond to the 10 different body parts, and the columns refer to (in order) y, x, α, respectively. A large number of example poses represented in this format will be provided to you for model learning.

There are two classes, humans and aliens, in the dataset. Humans and aliens have different body structures, therefore they display different pose variations. Our goal is to learn models that capture the variations of the poses in each of the two classes. The class labels are provided to you in the training set, so the problem is a typical example of supervised learning.

2.2 Variables and State Spaces

Let's consider a Bayesian network, where each node represents a body part. We denote the body part variable as $\{\mathbf{O}_i\}_{i=1}^{10}$, each of which take on continuous values in $\mathbb{R}^3 : \mathbf{O}_i = (y_i, x_i, \alpha_i)$[1]. Throughout the programming assignment, we assume y_i, x_i, and α_i are independent, so the probability of observing $\mathbf{O}_i$ is simply the product of the probabilities of observing y_i, x_i, and α_i

[1]You might notice that the domain of α is actually $[-\pi, \pi]$. Cutting the circle at any point might introduce artifacts in modeling. Therefore, for simplicity, in this programming assignment we will just assume that α can take on any real value and ignore the fact that α and $\alpha + 2\pi$ are actually the same.

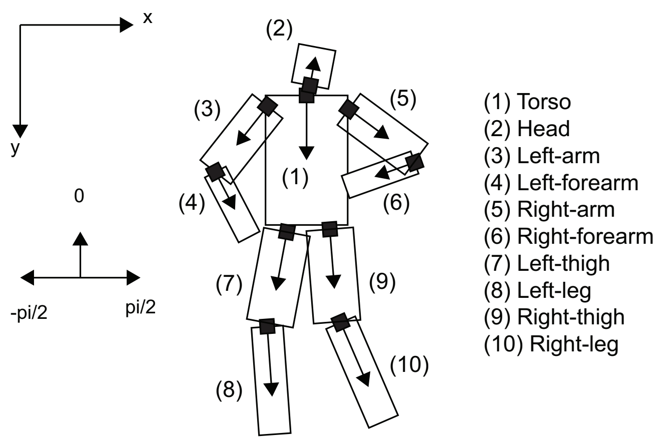


Figure 1: Representation of the body pose. Note that the boxes for the body parts are for visualization purposes only, and are not part of our model specification, i.e., we do not attempt to model the sizes of the body parts.

individually. We denote the class variable as $\mathbf{C}$, which takes on discrete values $\{1, 2\}$ for the two classes we have in this programming assignment, humans and aliens, respectively.

2.3 Naive Bayes Model

As a first attempt, we adopt the naive Bayes assumption: all variables $\{\mathbf{O}_i\}_{i=1}^{10}$ are independent given the class label $\mathbf{C}$. This assumption gives rise to the graph structure shown in Figure 2a. Recall that this is the plate representation we have encountered earlier in this course. In the naive Bayes model, each body part only has the class label as its parent. The variables can then be parametrized as follows,

$$P(\mathbf{C} = k) = c_k, \tag{1}$$

$$y_i|\mathbf{C} = k \sim \mathcal{N}(\mu_{ik}^y, {\sigma_{ik}^y}^2), \tag{2}$$

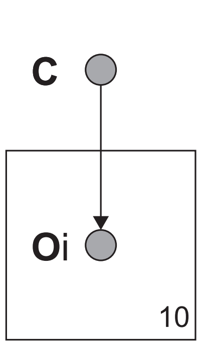
$$x_i|\mathbf{C} = k \sim \mathcal{N}(\mu_{ik}^x, {\sigma_{ik}^x}^2), \tag{3}$$

$$\alpha_i|\mathbf{C} = k \sim \mathcal{N}(\mu_{ik}^\alpha, {\sigma_{ik}^\alpha}^2), \tag{4}$$

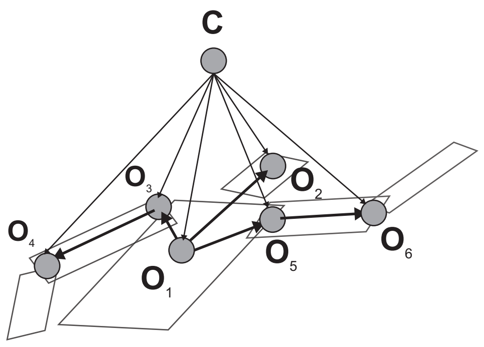
where $i = 1, 2, \ldots, 10$ indexes the body parts, and $k = 1, 2$ indexes the two classes. The notation $Y|X = x \sim \mathcal{N}(\mu, \sigma^2)$ means the conditional distribution of random variable Y given X is a Gaussian distribution with mean μ and variance σ^2. Given the class label, each continuous variable is modeled by a Gaussian distribution. Therefore each body variable will have two sets of Gaussian parameters, corresponding to the two classes. We will learn the parameters from the training data.

Since the class label $\mathbf{C}$ is always observed in the training data, we can collect the sufficient statistics for each of the classes from the training data.

- `FitGaussianParameters.m` **(5 points):** In this function, you will implement the algorithm for fitting maximum likelihood parameters $\mu = \mathbf{E}_{\hat{P}}[X]$ and $\sigma = \sqrt{\mathbf{Var}_{\hat{P}}[X]}$ of the



(a)



(b)

Figure 2: Graph structures. (a) The naive Bayes model. (b) Introducing skeletal edges among body part variables. For clarity we only show 6 upper body part variables.

Gaussian distribution given the samples. $\mathbf{E}_{\hat{P}}$ is the expectation with respect to the empirical distribution $\hat{P}$, which is specified by the dataset $\mathcal{D}$. The variance can be computed as $\mathbf{Var}_{\hat{P}}[X] = \mathbf{E}_{\hat{P}}[X^2] - \mathbf{E}_{\hat{P}}[X]^2$. Note that this function needs to return the standard deviation σ instead of the variance σ^2. You should fill in the code to compute the parameters. In this programming assignment we normalize the variance or the standard deviation by n as opposed to $n-1$, where n is the number of samples. That is, you need to set the option flag to 1 if you use `var` or `std` in Matlab and Octave.

2.4 Learning with Known Skeletal Structure

While the naive Bayes assumption results in simple model that is easy to learn, it makes unrealistic assumptions. It assumes that body parts are independent given the class label. But we know that is not the case, since body parts are connected by joints and therefore dependent based on how they are connected. Our second attempt will exploit the kinematic structure of the human body by adding edges among the body part variables.

Let's first assume that humans and aliens share the same skeletal structure, and that structure is known. Specifically, we take *torso* as the root variable, and *head*, *left-arm*, *right-arm*, *left-thigh*, *right-thigh* are its children nodes. We then add *left-forearm* as a child of *left-arm*, and *left-leg* as a child of *left-thigh*. Similarly, we add *right-forearm* as a child of *right-arm*, and *right-leg* as a child of *right-thigh*. Without the class variable $\mathbf{C}$, the body parts form a tree-structured model. Recall that a tree-structured model is a Bayes Net where each variable has at most one parent. With the class variable $\mathbf{C}$ as a parent to all the body part variables, the graph structure is shown in Figure 2b, where we only show $\mathbf{O}_1$ to $\mathbf{O}_6$ for clarity.

The next problem is how to parameterize the local CPDs. For the root body part $\mathbf{O}_1$ (*torso*), it only has the class variable $\mathbf{C}$ as its parent, so its parameterization is exactly the same as before. However, all the other body parts have two parents, the class variable and a parent body part. How should we handle this?

Earlier in this course, we learned about the conditional linear Gaussian (CLG) model to parameterize variables with both discrete and continuous variables as parents. Intuitively, for each assignment of the discrete parent, we have a different linear Gaussian model. In our case, we would learn a linear Gaussian model for humans, and a linear Gaussian model for aliens,

Formally, the configuration of each child body part $\mathbf{O}_i = (y_i, x_i, \alpha_i)$ is modeled as a linear

Gaussian in the configurations of its parent body part conditioned on each possible assignment of the class label:[2]

$$y_i|\mathbf{O}_{p(i)}, \mathbf{C} = k \sim \mathcal{N}(\theta_{ik}^{(1)} + \theta_{ik}^{(2)} y_{p(i)} + \theta_{ik}^{(3)} x_{p(i)} + \theta_{ik}^{(4)} \alpha_{p(i)}, {\sigma_{ik}^{y}}^{2}), \tag{5}$$

$$x_i|\mathbf{O}_{p(i)}, \mathbf{C} = k \sim \mathcal{N}(\theta_{ik}^{(5)} + \theta_{ik}^{(6)} y_{p(i)} + \theta_{ik}^{(7)} x_{p(i)} + \theta_{ik}^{(8)} \alpha_{p(i)}, {\sigma_{ik}^{x}}^{2}), \tag{6}$$

$$\alpha_i|\mathbf{O}_{p(i)}, \mathbf{C} = k \sim \mathcal{N}(\theta_{ik}^{(9)} + \theta_{ik}^{(10)} y_{p(i)} + \theta_{ik}^{(11)} x_{p(i)} + \theta_{ik}^{(12)} \alpha_{p(i)}, {\sigma_{ik}^{\alpha}}^{2}), \tag{7}$$

where $i = 2, \ldots, 10$ indexes the body parts, and $k = 1, 2$ indexes the two classes. The notation $p(i)$ denotes the parent body part of the ith body part.

Now consider the problem of learning the parameters in the CLG model. As before, the class labels are given. The only difference is that we now fit a linear Gaussian model instead of a Gaussian model for each body part variable (except the root body part, which only has the class variable as its parent). Let's now implement the learning process for conditional linear Gaussian models.

- **`FitLinearGaussianParameters.m` (15 points):** In this function you will implement the algorithm for fitting linear Gaussian parameters in the general form (such that it can be used in different scenarios):

$$X|U \sim \mathcal{N}(\beta_1 U_1 + \beta_2 U_2 + \cdots + \beta_n U_n + \beta_{n+1}, \sigma^2) \tag{8}$$

Note that we use β_{n+1} instead of the usual notation β_0 to be consistent with the indexing method of Matlab and Octave, and you should keep this in mind when you invoke this function in subsequent tasks. By taking derivatives of the log-likelihood function[3], we can see that the parameters $(\beta_1, \cdots, \beta_{n+1})$ satisfy the following system of linear equations:

$$\mathbf{E}_{\hat{P}}[X] = \beta_1 \mathbf{E}_{\hat{P}}[U_1] + \beta_2 \mathbf{E}_{\hat{P}}[U_2] + \cdots + \beta_n \mathbf{E}_{\hat{P}}[U_n] + \beta_{n+1} \tag{9}$$

$$\mathbf{E}_{\hat{P}}[X \cdot U_i] = \beta_1 \mathbf{E}_{\hat{P}}[U_1 \cdot U_i] + \beta_2 \mathbf{E}_{\hat{P}}[U_2 \cdot U_i] + \cdots + \beta_n \mathbf{E}_{\hat{P}}[U_n \cdot U_i] + \beta_{n+1} \mathbf{E}_{\hat{P}}[U_i] \tag{10}$$

where $i = 1, \ldots, n$. Now we have $n + 1$ variables and $n + 1$ equations, so we can use the backslash operator \ to solve them. More specifically, you can solve a linear equation $Ax = b$ by setting `x = A\b`. Having obtained the parameters $(\beta_1, \cdots, \beta_{n+1})$, the standard deviation σ can be computed by:

$$\sigma = \sqrt{\mathbf{Cov}_{\hat{P}}[X; X] - \sum_{i=1}^{n} \sum_{j=1}^{n} \beta_i \beta_j \mathbf{Cov}_{\hat{P}}[U_i; U_j]} \tag{11}$$

where the covariance can be computed as $\mathbf{Cov}_{\hat{P}}[X; Y] = \mathbf{E}_{\hat{P}}[X \cdot Y] - \mathbf{E}_{\hat{P}}[X] \cdot \mathbf{E}_{\hat{P}}[Y]$.

[2]In fact, the position of a body part doesn't have a linear dependency on the orientation angle of its parent body part, but rather the sine and cosine values of it. But for simplicity, we will use a linear model in this programming assignment.

[3]If you are interested in the details of the derivation, please refer to **17.2.4 Gaussian Bayesian Networks** of the text book.

3 Data Format and Code Structure

3.1 Data Structure

Before we start implementing all the learning procedures above, let's first take a look at the data structures we will use in this assignment. To simplify the interface of the program, we use a unified data structure to represent the parameters and graph structures we encountered in all previous subsections. Specifically, the parameters are encoded in a structural array P. The vector $P.c$ encodes the prior on class probabilities in Equation (1), where humans have class label 1 and aliens have class label 2. The graph parameterization is encoded in a 10×2 matrix G, where each row of G corresponds to a body part (in the order shown in Figure 1). The first column of G represents the graph parameterization choice:

- $G(i, 1) = 0$ indicates that body part i only has the class variable as its parent. In this case, $G(i, 2)$ can take on an arbitrary value. Its parameterization follows the naive Bayes model described in Equations (2,3,4). The parameters can be found in the following 1×2 vectors: `P.clg(i).mu_x`, `P.clg(i).mu_y`, `P.clg(i).mu_angle`, `P.clg(i).sigma_x`, `P.clg(i).sigma_y`, and `P.clg(i).sigma_angle`. For example, `P.clg(2).sigma_y(2)` is the standard deviation of the y-position of the head given that the class label is 2 (aliens).

- $G(i, 1) = 1$ indicates that body part i has, besides the class variable, another parent $G(i, 2)$, and the CPDs are parameterized following Equations (5,6,7). The parameters can be found in the 2×12 matrix `P.clg(i).theta` and 1×2 vectors `P.clg(i).sigma_x`, `P.clg(i).sigma_y`, and `P.clg(i).sigma_angle`. For example, `P.clg(i).theta(k, 9)` corresponds to $\theta_{ik}^{(9)}$ in Equation (7).

Later in the assignment, we will learn different graph structures for each of the two classes. In that case, a $10 \times 2 \times 2$ matrix G will be used, where $G(:, :, k)$ specifies the graph structure and parameterization for the kth class (in the same way as described above).

3.2 Provided Data

We provide you with all the data you need in a file `PA8Data.mat`, which contains the following data structures:

- `trainData`: This is a structure that contains two fields. The `data` field is a $N \times 10 \times 3$ matrix that contains N training examples. The `labels` field is a $N \times 2$ vector that provides the true class labels for the training examples. The entry in row i and column j is 1 if the ith example belongs to class j, and 0 elsewhere.

- `testData`: Similarly, the test data is provided in this structure. It has the same format as `trainData`. You will evaluate your learned models on this test set.

- `G1, G2`: We have constructed these 10×2 graph structures for you. `G1` is the graph structure for the naive Bayes model, and `G2` is the graph structure for the CLG model. They follow the graph parameterization choices as describes in 3.1.

To test your code, a separate file `PA8SampleCases.mat` contains the input and correct output for each of the sample cases that we test for in the submit script. For argument j of the function call in part i, you should use `exampleINPUT.t`$\#_i$`a`$\#_j$ (replacing the $\#_i$ with i). For output, look at `exampleOUTPUT.t`$\#_i$ `for the output to part` i`, or exampleOUTPUT.t`$\#_i$`o`$\#_j$ for output j if there are multiple outputs.

3.3 Infrastructure Code

The following functions are provided to you, and you do not need to modify them.

- `ShowPose.m`: This function outputs a bitmap image of the human figure given a pose configuration specified by a 10×3 matrix.
- `SamplePose.m`: This function takes as input the graph structure G and the parameter structure array P in the format specified above, samples from the distribution and outputs a pose configuration represented by a 10×3 matrix. You can also fix the class label and sample from the conditional distribution. It calls `SampleGaussian.m` and `SampleMultinomial.m`, which are as simple as their names suggest.
- `VisualizeModels.m`: This function gives you an intuitive assessment of the quality of your learned model by visualizing it. The code calls `SamplePose.m` and `ShowPose.m`, and shows a window with N sub-windows. Each sub-window shows a series of samples from the model for each of the N classes.
- `VisualizeDataset.m`: This function shows each example pose in a dataset one-by-one in an animation.
- `MaxSpanningTree.m`: This function finds the maximum spanning tree given the symmetric weight matrix. Output of this function is represented as an (non-symmetric) adjacency matrix A (where one of $A(i, j)$ and $A(j, i)$ is 1 if i and j are connected by an edge, both are 0 otherwise).
- `ConvertAtoG.m`: Convert the adjacent matrix representation into our representation G of the tree structured model. In this function we always use the first body part (*torso*) as root node (such that the conversion is unique), and we always use the parameterization in (5,6,7) (by setting $G(i, j) = 1$ for $i \neq 1$).
- `lognormpdf.m`: This function computes the Gaussian probability density function in log space at the input value, given the mean and standard deviation.
- `GaussianMutualInformation.m`: This function computes the mutual information between two multi-dimensional Gaussian variables (they can have different dimensionality). You might find it useful when learning the graph structure.

4 Model Learning

We are now ready to implement the learning procedures. You will subsequently implement and experiment with different model choices. Let's first load the data that we will be working with, and visualize the dataset by running the following,

```
load PA8Data.mat;
VisualizeDataset(trainData.data);
```

You should see an animation of humans and aliens with different poses. To evaluate how well the models are capturing these pose variations in the provided data, we first implement the following function.

- `ComputeLogLikelihood.m` **(20 points):** This function computes, given the model P, coupled with the graph structure G, and the dataset, the log likelihood of the model over the dataset:
$$\sum_{i=1}^{N} \log P(\mathbf{O}_1 = \mathbf{o}_1^{(i)}, \cdots, \mathbf{O}_{10} = \mathbf{o}_{10}^{(i)}) \tag{12}$$
where N is the number of examples in the dataset. The log function here refers to the natural logarithm function, which we use throughout this programming assignment. The likelihood of each individual observed pose configuration $\{\mathbf{O}_i\}_{i=1}^{10}$ can be computed as follows.
$$P(\mathbf{O}_1, \cdots, \mathbf{O}_{10}) = \sum_{k=1}^{2} P(\mathbf{C} = k, \mathbf{O}_1, \cdots, \mathbf{O}_{10}) = \sum_{k=1}^{2} P(\mathbf{C} = k) \prod_{i=1}^{10} P(\mathbf{O}_i | \mathbf{C} = k, \mathbf{O}_{p(i)}). \tag{13}$$
where we sum over the joint probabilities of each possible class label. Recall that $\mathbf{O}_{p(i)}$ is the parent of $\mathbf{O}_i$, and $p(i)$ can be found in the given graph structure $G(i, 2)$. While the Matlab/Octave function `normpdf` can be used to evaluate the Gaussian density function, numerical issues introduced by extremely small numbers returned by `normpdf` can cause problems. Therefore, instead of multiplying probabilities, we will take the logarithms of the probabilities and sum the log-probabilities. You function should compute everything in log-space, and return log-probabilities instead of probabilities. You may find the function `lognormpdf.m` we provided useful. Notice that your code should handle the case where we have multiple graph structures for each class. That is, G can be either 10×2 (shared structure) or $10 \times 2 \times 2$ (different structure for each class), and your function should be able to choose the right graph based on the dimension of G. Please follow the instructions in the comments in this file to complete the implementation. You may refer to the usage of this function in `LearnCPDsGivenGraph.m`.

The log likelihood of the model over the dataset measures how the model fits the observed data. This measure allows us to compare different model choices quantitatively.

4.1 Learning with Known Graph Structures

We are now ready to learn the model parameters. You should write code to fit the model given the graph structure.

- `LearnCPDsGivenGraph.m` **(25 points):** This is the function where we learn the parameters. This function takes the dataset and a known graph structure G as input, looks into G to decide which parameterization is used for each of the body parts. So this function handles all the model choices we discussed in a unified interface. We have already written some infrastructure code in this function for you, and all you need to do is to invoke the functions you just implemented by feeding them with the correct parameters, and write the output in the correct format into the parameter structure array P. Please follow the instructions given as comments in this file to complete the implementation. You may look into the sample output `exampleOUTPUT.t4o1`, which contains the model P learned for the given graph `exampleOUTPUT.t4a2`, to understand the structure of P. When learning the parameters, you may need to pass a subset of the dataset, which is a 3-dimensional matrix, to `FitLinearGaussianParameters.m`, which takes a 2-dimensional matrix as its second argument. In this case, you may find the command `squeeze` or `reshape` helpful for removing singleton dimensions in a matrix.

We evaluate the learned models on the test data provided. Fill in the following function:

- `ClassifyDataset.m` **(15 points):** This function takes a dataset, a learned model P, and a graph structure G. It then predicts the class labels for all test examples in the dataset, and computes the accuracy of the model on the entire dataset. To perform the classification task, you need to find the most likely class label assignment for each test example, i.e., the class label that has the highest joint probability with the observed instance. The accuracy is then computed as {number of correctly classified instances}/{total number of instances}.

Now, you can learn the model parameters for the difference model choices. In particular, run the following,

```
[P1 likelihood1] = LearnCPDsGivenGraph(trainData.data, G1, trainData.labels);
```

which learns the parameters for the naive Bayes model. We can run the classification task for the naive Bayes model you just learned,

```
accuracy1 = ClassifyDataset(testData.data, testData.labels, P1, G1);
```

How does the naive Bayes model perform? Now let's visualize the learned model by sampling from the learned parameters,

```
VisualizeModels(P1, G1);
```

Now it should be clear why the model is not performing so well – body parts are disconnected in this model. Now let's try to improve the model by putting the CLG model to work:

```
[P2 likelihood2] = LearnCPDsGivenGraph(trainData.data, G2, trainData.labels);
```

Repeat the classification task for the new model, `(P2, G2)`. Compare the log-likelihood scores and the accuracies. You should be able to see a higher likelihood score and a better accuracy for the new model. If your implementation is correct, you should get accuracies of 79% and 84% respectively. Visualize the model, and see where the improvement is coming from. The structure for humans should now be clearly seen.

4.2 Learning Graph Structures

In our previous attempt, we tried to learn model parameters for humans and aliens with the same skeletal structure. When visualizing the models, you should be able to see that the model captures the human body very well, but fails on aliens. Some body parts are detached from the rest of body, and the body structure is not clear. This is because we are using a graph structure different from the true skeletal structure of the aliens. Therefore, instead of fitting the data to a given structure, we will try to learn the structures from data.

We want to learn a tree-structured model for each class. Given a scoring function that satisfies decomposability and score-equivalence, we can compute the score, or weight, between all pairs of variables, and find the maximum spanning tree. Using the likelihood score, we set the edge weights to be:

$$w_{i\to j} = \text{Score}_L(\mathbf{O}_i|\mathbf{O}_j : D) - \text{Score}_L(\mathbf{O}_i : D) = M \cdot \mathbf{I}_{\hat{P}}(\mathbf{O}_i, \mathbf{O}_j) \tag{14}$$

where M is the number of instances and $\mathbf{I}_{\hat{P}}$ is the mutual information with respect to distribution $\hat{P}$. We have learned in class the definition of mutual information between discrete variables and how to compute it. It turns out that the same concept applies to continuous variables as well, and it has a closed form expression for Gaussian variables. In general, the mutual information between two multidimensional Gaussian variables $\mathbf{X}$ and $\mathbf{Y}$ can be computed by:

$$I(\mathbf{X}, \mathbf{Y}) = \frac{1}{2}\log\left(\frac{|\Sigma_{\mathbf{XX}}| \cdot |\Sigma_{\mathbf{YY}}|}{|\Sigma|}\right) \tag{15}$$

and

$$\Sigma = \begin{pmatrix} \Sigma_{\mathbf{XX}} & \Sigma_{\mathbf{XY}} \\ \Sigma_{\mathbf{YX}} & \Sigma_{\mathbf{YY}} \end{pmatrix} \tag{16}$$

where $|\cdot|$ denotes the determinant of a matrix, and Σ denotes the covariance matrix.

Now we are set to use it to learn the graph structure for each class. This is done in the following functions that you will implement.

- `LearnGraphStructure.m` **(10 points):** Learning the tree-structured graph from data should be quite straightforward given the two functions `GaussianMutualInformation.m and MaxSpanningTree.m` we provide. You need to fill in the code to compute the weights between every pair of body part nodes. Computing the maximum spanning tree from the weights is already implemented for you. Follow the instructions given in the code to complete the implementation.

- `LearnGraphAndCPDs.m` **(10 points):** This function learns the parameters, as well as the graph structure, for each class of data. For learning the graph structure, you should call the function `LearnGraphStructure.m` you just implemented, and use the provided function `ConvertAtoG.m` to convert the maximum spanning tree to the desired graph structure G. You can copy most of the code from `LearnCPDsGivenGraph.m`. The only difference is that instead of using any of the provided graph structures, we estimate the graph structure for each of the classes.

When you are done, try:

```
[P G likelihood3] = LearnGraphAndCPDs(trainData.data, trainData.labels);
ClassifyDataset(testData.data, testData.labels, P, G);
```

If your implementation is correct, your model should achieve 93% accuracy! Compare the log-likelihood score with previous results. Visualize the learned graph structure for aliens, and you will see the body structure of the aliens from the samples of the model.

5 Conclusion

Congratulations! You have successfully used graphical models to learn classifiers to distinguish between two poses drawing from different distributions and graph structures. Even though it is a simple, synthetic dataset, we still saw how one can improve the models by incorporating knowledge about the dataset. In the next programming assignment, you will extend what you have done for this assignment to design models for activity recognition on real data.