B553: Probabilistic Approaches to Artificial Intelligence

Assignment 3 – Project Report

Nasheed Moiz (nmoiz) and Chaitanya Bilgikar (cbilgika)

Learning

For the learning part of the assignment, the starter code provided a method called load_groundtruth, which takes in a file name (gt.train for learning) and populates a vector of GTImage with all the relevant information for us. We have used one method to do the learning. This method is called from the main. It has 3 parameters namely, the vector of GTImage called training_data which is actually generated by load_groundtruth, a pointer to a 6*6 2D array of Coordinates for storing means, and another pointer to a 6*6 2D array for storing variances. The 2 2D arrays are allocated in the same scope that learning is called from i.e. main. After learning is done running, these 2 2D arrays should have what they are supposed to.

It is worth noting that even though the 2D arrays have a capacity of 36, every slot in this array will not be full. We will be storing means and variances of all possible combinations (2 at a time) of the 6 facial parts for a total of 6choose2 or (5+4+3+2+1) = 15 elements. The indexes in the arrays are used to access the mean or variance for any two facial parts. The convention we have used is that for instance for parts nose (2) and chin (5), we will store the mean and variance respectively in index [2][5] of the 2 2D arrays. For left eye(0), and right mouth(4), we can access their mean and variance by using the index [0][4]. Notice the convention we are using that out of the two parts, the part with lower index is simply used first, ie for the row index.

The calculation of these values involves iterating through the training data. In each iteration of the training data we have a for loop like so:

```
for(i=0; i<6; i++)
    for(j=i+1; j<6; j++)
```

which allows us to cover all 15 possible combinations of facial parts.

- The definition of the mean of left eye and right eye for instance is sum of observations divided by size of training data.
- Here each ‘observation’ is a coordinate object, where its row value stores the squared difference of the row value of two parts and its column value stores squared difference of the column value of the two parts for any single entry in the training data.
- For the variance, we iterate through training data again for all 15 combinations, this time calculating squared difference of each ‘observation’ and its corresponding entry in mean. We sum these for each entry in training data, and divide by training_data.size(). In the end, the 2D array vars at index [1][3] should have a coordinate object with a row and a column value, which is the variance of parts right eyes and left mouth.

MAP Inference using Max-Product Belief Propagation

For our implementation of exact MAP inference we have written a method called Phi_calculator which comes up with two distributions of size rows*columns. As we know a Phi value is calculated for two parts, where one of them is considered a parent because we assume

we are working with a tree. So if we consider model b) where left eye is parent and all other parts are children. Our method's first distribution called Phi_lefteye_righeye stores potential values where in index [4][25] for instance we store for parent (left eye's) coordinates being (4,25) the maximized potential over all possible coordinates of its children (left eye). The second distribution in a similar fashion stores for each possible coordinate value of the parent, the most likely coordinate of the child.

For details about how the potential is calculated, please refer to code and comments. It is worth noting that each time we calculated a potential value using formula 5 in assignment document, we add the unary potential of the child on the spot. This constitutes the message sent by the root to its children in a sense.

This Phi calculator method does the bulk of the computation. In addition we have 3 different methods namely MAP_Inference_B, MAP_Inference_C and MAP_Inference_D that for the three respective graphs creates appropriate variables, and calls Phi calculator with appropriate parameters.

Let us do a quick walkthrough of an example, for model B.
Model B, left eye is root and all others are leaves or its children.

We declare the following variables:

Vector of vector of coordinates left_eye&right_eye = will store for each index, where index signifies coordinates of left_eye ie the root, the most likely coordinate for its child. Similarly, we have 4 other variables left_eye & nose, left_eye & left_mouth, left_eye & right_mouth, and left_eye & chin.

double plane left_eye&right_eye = will store a distribution that is the pairwise potentials of the two variables plus the unary of the child, ie right eye.
Similarly we have 4 other doubleplane variable namely, left_eye & nose, left_eye & left_mouth, left_eye & right_mouth, and left_eye & chin.

Now, we declare a double plane left_eye, where we store in any given index the sums of the past 5 variables + the unary of the left_eye. From this double plane, maximizing gives us the assignment for left_eye ie the root.

Using the assignment for the root ie left_eye, we can extract assignments for the child nodes by using the x and y value of left_eyes coordinates as indexes in the 5 vector of vectors. See code for more details.

Loopy BP

In this part each node updates the belief states of all its neighbors based on:

- Belief states of all its other neighbors.
- Its own unary potential
- Pairwise potential between this node and its parents.

Now each node maintains two belief states, one for time t and another for time t+1 (denoted as node_message_old and node_message respectively in the code). When each node updates the

belief state of another node, it uses its belief state at time t to update the belief state at time t +1 for the neighbor. We perform this update for every node in the graph. Once all the nodes have updated the belief states of all their neighbors, we replace the belief states at time t by the belief states at time t+1 (we do this every node). We repeat this loop a set number of times. Now this number of repetitions is what put us in a bit of a bother. We were not sure as what would be the optimum number of times to repeat this loop. For some images, the results improves as the number of repetitions was increased and for some it made not much of a difference.
At the end of the loop, we are left with the node_message tables for every node in the graph. To get the best possible assignment of pixels for the parts, we then perform an argmax over this table for every node. The row and column pair that has the maximum value is taken as the best possible assignment for that particular part.

Note: Accuracy for each method is being computed on the fly and being shown as an output of the program.

References: Consultations with Debpriya Seal, Shrutika Poyrekar, Zoujin Ouyang. Online videos from Daphne Koller's free YouTube Channel.