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 method evaluation where most of our calls our made. 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 this 2D array stores the means and variances such that if we access mean [LEFT_EYE][NOSE] we will get the value calculated by learning. This value is useful when trying to send a message from left eye to nose. However when we want to send a message for instance from nose to left eye, we will pass the negative of mean [LEFT_EYE][NOSE]. Hence accessing means [NOSE][LEFT_EYE] will give us the negative of the value returned by the indexes exchanged. The variances will be the same for both instances since they are squared and always positive. We have only calculated the positive values and in the end of the method called learning, in a for loop stored the negative values in the indexes reversed as described above. We are doing this because in our message calculator called Phi_calculator, where we use the Gaussian formula, we cannot specify which part is sending a message to which other part. This is a problem because the formula requires us to do (Li.row - Lj.row) and (Li.col - Lj.col). So we simply modify the means appropriately to get the pairwise potential that we want.

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 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.

- Note: We have defined some constants at the beginning of the file that represent the integer values for each part (e.g. 0 for Left Eye, 1 for Right Eye etc.). Also the variable STEP_COUNT is used to denote that number of pixels that we scan for every image for the calculation for the pair wise potentials.

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 in the image. This method actually constructs messages and stores them as SDoublePlane variables since we have not used an explicit tree data structure.

As we know a message is calculated for two parts, where one of them is considered as the sender and the other the receiver. So if we consider model b) where left eye is sender and all other parts are receiver (Other parts only send unary to left_eye so we don't need to compute those). Our method's first declared SDoublePlane variable 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 the receiver (right eye). The second distribution (called SDoublePlane coord_map_le_re in our code) in a similar fashion stores for each possible coordinate value of the sender (left_eye), the most likely coordinate of the receiver (right_eye).

This way left_eye can maximize and get an assignment for itself. Using its assignment, we can look up the assignments for the other parts in the respective coord_map variables as described above.

The 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.

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.

Evaluation

The method evaluation is called from the main. Inside it we call learning, all 4 kinds of inferences each of which return a GTImage with their assignments for facial parts for each image in testing set. Then using the testing truth vector of GTImages, we compare and increase counters which are in the end divided by total number of images * 6, giving us an accuracy value for each of the 4 kinds of inferences.

Accuracies

We calculated the accuracies for all the inference techniques using a step count of 4 for all the methods (we did this as a lower step count results in a large execution time). We tested each method on all the files in the gt.test file. Shown below is the accuracy for each technique. A better accuracy can be achieved with a lower step count.

Model B	Model C	Model D	Model E (loopy)
43.33%	27.8%	40%	15%

We have discussed this assignment (on a high level) with Debpriya Seal and Shrutika Poyrekar.