CSE 518 HCI Assignment - 2 Report
114707888, Shiva Sankeerth Reddy Yarradla

Task:

Given a dictionary containing 10000 words, decode a user input gesture using the SHARK2 algorithm and output the best-decoded word.

Implementation:

I implemented the SHARK2 algorithm by completing the definition of the following functions:

1) **generate_sample_points(points_X, points_Y)**:
    a) I defined this function to convert every gesture or template to a set of 100 points, such that we can compare the input gesture and a template computationally.
    b) First, I calculated the difference between consecutive X and Y coordinates.
    c) Then, I used the above differences to calculate the euclidean distance between the points by applying a square root function to the above differences. Then I calculated the cumulative distance.
    d) The total distance is the last element of the cumulative distance array. I used the total distance to normalize the cumulative distances.
    e) Then I used the normalized cumulative distances and the X and Y points to define two linear interpolation functions.
    f) The final 100 sample points are obtained using the above interpolation functions on 100 evenly spaced points between 0 and 1.
2) **do_pruning(gesture_points_X,gesture_points_Y,template_sample_points_X,template_sample_points_Y)**:
    a) I defined this function to prune the words so as to narrow down the number of valid words so that ambiguity can be avoided.
    b) I have set a pruning threshold of 30.
    c) I then created a NumPy array for gesture-start and gesture-end points for enabling faster computation.
    d) I gathered the start and end points for each template into a NumPy matrix.
    e) Then I calculate the distance between the start and end points of both gesture and template points.
    f) Then I gathered the indices whose start + end distances are less than the threshold. These are our valid indices.
    g) These valid indices are used to get the X and Y coordinates of valid template sample points, valid words, and their respective probabilities.
    h) The above are returned along with the valid indices to help us with other tasks.

3) **get_shape_scores()**:
    a) I defined this function to compare the sampled input gesture (containing 100 points) with every single valid template (containing 100 points) and give each of them a shape score.
    b) First I have set my length L value to be 200. We now start out by scaling the gesture points. For this, we calculate the width and height by taking the difference between the minimum and maximum of X and Y coordinates respectively. Now we get the scaling factor using the formula, **S = L / (max(Width, Height,1))**
    c) Now, we multiply the gesture points with the scaling factor 'S' to get the scaled gesture points.
    d) I then translated the coordinates around the centroid of the gesture.
    e) I then calculated the Euclidean distance between the pre-computed normalized Template points and gesture points.
    f) The shape scores are obtained by summing the above euclidean distances and dividing them by the number of points, 100 in our case.
4) **get_location_score()**:
    a) I defined this function to compare the sampled user gesture (containing 100 points) with every single valid template (containing 100 points) and give each of them a location score.
    b) I followed the implementation of the Location Recognition Channel presented in the given paper. Calculated this algorithm after the pruning step.
    c) Initialized location scores NumPy array with zeros, and also gesture_points array with gestures sample points.
    d) According to the paper, we need to compute the euclidean distance of gesture input and templates for each word, calculated euclidean distance between valid gesture points and valid template points.
    e) I then found the distance of the closest gesture point to each template point and vice-versa.
    f) Compared if minimum distance is greater than radius for all possible samples.
    g) If the minimum distance is greater than the radius, calculate the location score for each valid word using the pre-computed alphas and delta.
    h) The final location_scores is returned and is used in the calculation of the integration score.
5) **get_best_word(valid_words, integration_scores)**:
    a) I defined this function to select top-n words with the highest integration scores and then use their corresponding probability (stored in variable "probabilities") as weight. The word with the highest weighted integration score is exactly the word we want. We follow the same process but instead of multiplying the highest integration scores with corresponding probabilities, we multiply the lowest integration scores with (1 - probabilities).
    b) I have selected my **n** to be 4, to get the top 4 words.
    c) The final score for these words is calculated using this formula:
        i) Final_score(x) = integration_score(x)*(1-word_probability(x))
    d) The word with the lowest final score is selected as the best word.
    e) The rest of the top words are given to the user as suggestions.

6) **get_integration_scores(shape_scores, location_scores)**:
    a) After we have computed both the shape_scores and location_scores, we get our integration score by using this following weighted addition:
        i) **Integration score = (shape_coeffiecient * shape-score**) + (**location_coeffiecient * location score**)
    b) I have taken the **shape_coeffiecient** to be 0.2 and **location_coeffiecient** to 0.8.
    c) The resulting integration score is returned which is used to get the best word.

Other **Important and Noticeable** code changes to make the whole server respond faster:

1) I have pre-computed the alphas using the method mentioned in the given paper. I have done this so that each time the gestured is inputted, we don't need to compute this constant array. This change has massively improved the performance of the system.
2) After generating the template sample points, I have scaled, normalized, and translated these points around their centroid. I have done this to reduce the computation time every time a gesture is inputted.
3) I have used the above pre-computed normalized template points in the get_shape_score function using the valid_indices computed in the do_pruning function. This has given a performance boost to the server.