**REPORT**

1. We first parsed the 10,000 words from the words.txt file, and stored the words and their corresponding probabilities in a dictionary (HashMap) **probabilities**. We stored all the words in a list **words**. Probabilities dictionary is used to filter out the suggested words later in the code.
2. When the application is run, a QWERTY keyboard will be displayed at **﻿**[**http://127.0.0.1:5000/**](http://127.0.0.1:5000/)**.**
3. A swipe on the keyboard will generate X and Y coordinates of the swipe, and are stored in **gesture\_points\_X** and  **gesture\_points\_Y.**
4. We use these points to sample 100 points.
   1. Generate\_sample\_points():
      1. Arguments: gesture\_points\_X, and gesture\_points\_Y (The coordinates to be sampled)
      2. (Reference: ﻿https://stackoverflow.com/questions/52312513/numpy-fast-regularly-spaced-average-for-large-numbers-of-line-segments-points)
      3. Sampled 100 points from these points. Normalized using cumulative sum of Euclidean distance.
      4. Used interp1d from numpy and sampled 100 points (Ref: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html>)
5. Now we prune the gesture and compare with the templates to find the best possible words.
   1. Do\_pruning():
      1. Arguments: gesture\_points\_X, gesture\_points\_Y, template\_sample\_points\_X, template\_sample\_points\_Y.
      2. We extract the starting and ending coordinates of the gesture from gesture\_points\_X, and gesture\_points\_Y.
      3. We calculate the distances between the words and our gesture.
      4. When the distance is less than the threshold, we add that particular word to our list of **refinedWords** and we return it to be used in get\_shape\_scores
6. Now we determine the shape\_scores for the filtered refinedWords obtained from the pruning method.
   1. Get\_shape\_scores():
      1. Arguments: ﻿refinedIdxes, gesture\_sample\_points\_X, gesture\_sample\_points\_Y, valid\_template\_sample\_points\_X, valid\_template\_sample\_points\_Y.
      2. We calculate the scaling factor using the formula,
         1. S = L / Max(W, H)
      3. We calculate the distance and divide it by 100 for every possible word and generate the Shape Score.
7. Now we calculate the location scores for the refinedWords obtained from the pruning method.
   1. Get\_location\_scores():
      1. Arguments: ﻿gesture\_sample\_points\_X, gesture\_sample\_points\_Y, valid\_template\_sample\_points\_X, valid\_template\_sample\_points\_Y.
      2. In this we calculate the location score with respect to all the words obtained from the pruning step (refinedWords).
      3. We extract the points from the gesture.
      4. For each point we find the closest template and gesture points and check if they are greater than the defined radius. If true, we construct **deltas** using the Euclidean distances and multiply with **alphaList** which was determined earlier.
      5. Else we take it as zero.
      6. We append this to the list locationScores and return, so that this list can be used in further calculating the integration scores.
8. After obtaining the shape, and location scores, we proceed to calculate the integration scores.
   1. Get\_integration\_scores():
      1. Arguments: ﻿shape\_scores, location\_scores
      2. We calculate the integration scores using the formula,
         1. Integration score = (a \* Shape score)+ (b \* Location score)
         2. Where a + b = 1
         3. I used a = 0.4, and b = 0.6.
         4. I tried different pairs of values like (0.2, 0.8), (0.25,0.75), (0.3, 0.7), (0.35, 0.65), (0.75, 0.25) and so on. It worked best with a less than b for me.
      3. We return the integrationScores so that we can extract the best words from all the filtered words.
9. Probabilities of the words being used plays a major role in suggesting the words to a user. Some words like **of, and off** have the same gesture. We eliminate such ambiguities using the probability of the words.
   1. Get\_best\_word():
      1. Arguments: ﻿valid\_words, integration\_scores
      2. We filter the words further on the basis of integraition scores
      3. We only take the best n words, I took n = 3.
      4. We sort the words based on the integration scores and take the first n words.
      5. We calculate the FinalScore using the formula,
         1. FinalScore = IntegrationScore \* (1 – probability)
         2. We get the probability of the word from the probabilities dictionary that we determined in the first parsing step.
      6. We iterate through all the scores, and return the word which got the best score (That is the minimum score).
10. We will get the bestWord from the get\_best\_word() method.
11. A timer which has started earlier will now end, and the total time taken in decoding along with the word is now displayed on the web page.

Example output:Graphical user interface

Description automatically generated

A picture containing application

Description automatically generated

A picture containing graphical user interface

Description automatically generated