DESIGN DOCUMENT

Our task was started by first cleaning the given corpus (output as *CleanDutch.txt and CleanEnglish.txt*) and extracting the distinct words in both the languages (output as *DistinctDutch.txt and DistinctEnglish.txt*) with the help of **FinalFileSplitter.py** module with the help of various methods described below: -

1. cleanEnglish and cleanDutch: At the start some data cleaning was require to be done as the document contained many special characters such as "**,**","**/**","**!**","**?**" etc. which are included in the list ***"I".*** These methods take every line of the documents English_Updated.txt and Dutch_Updated.txt and check every string in the lines and if they contain a special character they append the string with a " " (whitespace) which would later be used to split them into two different strings(made up of words) and these methods return a list of new lines in the form of **finalEnglish** and **finalDutch** lists which contain all the lines in lowercase and all the special characters removed.

2. readCorpus and extractWords: These methods extract all the words or strings in the corpus and puts them in their respective lists.

The **FinalProject.py** has the IBM model 1 and the innovated model and a driver function to translate a query into a language of choice. The various methods used in this module are:

1. initial_translation_probs: Given a corpus this method is used to generate the first set of translation probabilities, we assume that for a word in English and set of words in Dutch it is equally likely that the English word will translate to any word in Dutch.

2. euclidean_distance: This method calculates the Euclidean distance between the 2 vectors where the two of them are identical in structure

3. has_converged: This method concludes the model when in the final two iteration 'prev' and 'curr' has converged i.e. when the distance is finally less than the threshold set by us.

4. train_iteration: This method performs one iteration of the EM-Algorithm, where 'to_train' is the corpus object to train from, 'total_s' is the count of the destination words weighted according to their translation probabilities and 'prev_prob' is the translation probabilities from the last iteration of the EM algorithm.

5. model1: This method trains the translation model on the given corpus and threshold value iterating till the probabilities converge (using IBM model 1 with EM algorithm). It first reads the corpus using **readCorpus method** then creates a **list of dictionaries** with the English sentence mapping to its corresponding Dutch sentence and vice versa. Then we iterate the initial translation probabilities using the **train_iteration method** and return 2 dictionaries with English word as key and its Dutch translation as value and vice versa in **res** and **res1** respectively.

The following methods are used in our innovated model:

6. find_commom_lines: For every word in English and Dutch this method **returns** a dictionary(**dict_en,dict_nl**) with the words as **key** and a list containing the sentence numbers of the sentence containing the given word as **value**.

7.find_mappings: For each word in English this method find lines one by one that contain the given word and keep iterating until we get the lines that only contain our word as common so that when we use **model1_local method** our probability converge giving the output as the correct translation of our word in that instance of the loop and we do the same for Dutch words.

8. runDemoStuck: This method runs the whole program from the start and has the driver function

9.runDemoSmooth: This method loads the pickle files (dict_en and dict_nl) which contain the already runs and mapped dictionaries for English to Dutch translation and vice versa and has the driver function

The **metrics.py** has functions for evaluating the precision of translation using cosine similarity and Jaccard coefficient.

Algorithm Used: A statistical model has been trained for alignment and translation using IBM model I and Expectations Maximization (EM) algorithm as discussed in the class where all the probabilities would converge after a certain amount of iterations to a value less than or equal to the threshold value set.

DESIGN DOCUMENT

Input: set of sentence pairs $(\mathbf{e}, \mathbf{f})$
Output: translation prob. $t(e|f)$

initialize $t(e|f)$ uniformly
while not converged **do**
  // initialize
  count$(e|f) = 0$ **for all** e, f
  total$(f) = 0$ **for all** f
  for all sentence pairs (e,f) **do**
    // compute normalization
    for all words e in e **do**
      s-total$(e) = 0$
      for all words f in f **do**
        s-total(e) += $t(e|f)$
      end for
    end for
    // collect counts
    for all words e in e **do**
      for all words f in f **do**
        count$(e|f)$ += $\frac{t(e|f)}{\text{s-total}(e)}$
        total(f) += $\frac{t(e|f)}{\text{s-total}(e)}$
      end for
    end for
  end for
  // estimate probabilities
  for all foreign words f **do**
    for all English words e **do**
      $t(e|f) = \frac{\text{count}(e|f)}{\text{total}(f)}$
    end for
  end for
end while

<u>Limitation</u>:

1. IBM model 1: The main limitation we faced when using the model was that it used up too much RAM for performing the translation. The other limitation was regarding the reordering and adding or dropping of words as in most cases the words that follow each other in one language would have a different order after translation but we could implement this in our model due to the limitation of the computing power.

2. Innovated model: This model has the limitation that it couldn't differentiate between the alignment for the correction tranlation