# DESIGN DOCUMENT

The IBM Model 1 is implemented in the project to create a translation engine. The translator can convert sentences or documents from English to Dutch and vice-versa. The project consists of a README.md file, Main.py, TrainModels.py, TestModels.py, IBMModel1.py, EnglishtoDutch.py, DutchtoEnglish.py, CosineSimilarity.py, Common.py and Jaccard.py.

The given dataset is divided into training and test sets. The training set is used to train the model while the test set is used to test the accuracy of the model.

The design document provides a detailed explanation of the algorithm used, implementation of the algorithm in Python, control flow of the program and the various limitations associated with the algorithm and its implementation.

## ALGORITHM

The EM(Expectation-Minimization) algorithm is an important tool which is used in the translation process. The EM algorithm helps in maximizing a given likelihood function. We can assume that the given dataset is derived from a mixture of models each of which has its own set of parameters. An initial assumption is made about the values of the parameters of each model. The expectation step of the algorithm calculates the probability of an object that it belongs to a particular model. This step is done for each model. This is succeeded by the maximization step which, based on the previously calculated expectations of the objects, recalculates the parameters of each model while simultaneously maximizing the likelihood function. At the end of the procedure, what we obtain are clusters, each one belonging to a particular model of which the parameters are known. The given steps can be performed until the user's discretion. They can be controlled by using a fixed number of iterations, limiting the change in the values of the parameters or if the data can be visualized until satisfactory clustering has been observed.

The EM algorithm finds its implementation in the IBM Model 1. The model stores the probabilities of each word in a language being the direct translation of each word in the second language. The model first goes through the corpus to create a list of the words in each language and a matrix is created where each cell(i,j) corresponds to the probability of the word I being a translation of the word j. This matrix is populated when the model reads the corpus again. The two languages used here are English and Dutch. The model takes in one sentence and its translation at a time. It generates all possible alignments from that sentence and calculates the probabilities of the alignments, given the foreign sentence. An alignment identifies which English word each Dutch word originated from. Formally an alignment a is $\{a_1,a_2 \ldots a_m\}$ where each $a_j \in \{0,l\}$. The use of EM algorithm in IBM Model 1 is done in the following manner:

- Expectation step where the model is applied to the data. Here, the mixture of models are the various alignments and the parameters associated with the individual models are the possible values of the alignments.

- Maximization step where the assigned values in the above step are considered as facts, recalculate the probabilities of the alignments and assign those values as the model parameters.

We define the terms used further:

a: An alignment

$p(e, a|d) = \varepsilon / (l + 1)^m * \Sigma\, t(e_j \mid d_{a(j)})$

Here, e is an English sentence, d is a Dutch sentence, l is the number of words in the Dutch sentence, m is the number of words in the English sentence and epsilon is a normalization constant. $t(e_j \mid d_{a(j)})$ is the probability that the English word is derived from the $j^{th}$ word and this is summed for all words in the English sentence.

We need to calculate p(a | e,d) which implies the probability of a particular alignment given the English and the Dutch sentence. Applying the chain rule,

$p(a \mid e,d) = p(e,a \mid d) / p(e|d)$

We already have the formula for $p(e, a|d)$.

We need to compute $p(e|d)$.

$$\begin{aligned}
p(\mathbf{e}|\mathbf{f}) &= \sum_{a} p(\mathbf{e}, a|\mathbf{f}) \\
&= \sum_{a(1)=0}^{l_f} \cdots \sum_{a(l_e)=0}^{l_f} p(\mathbf{e}, a|\mathbf{f}) \\
&= \sum_{a(1)=0}^{l_f} \cdots \sum_{a(l_e)=0}^{l_f} \frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)})
\end{aligned}$$

$$
\begin{aligned}
p(\mathbf{e}|\mathbf{f}) &= \sum_{a(1)=0}^{l_f} \dots \sum_{a(l_e)=0}^{l_f} \frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) \\
&= \frac{\epsilon}{(l_f+1)^{l_e}} \sum_{a(1)=0}^{l_f} \dots \sum_{a(l_e)=0}^{l_f} \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) \\
&= \frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} \sum_{i=0}^{l_f} t(e_j|f_i)
\end{aligned}
$$

$$
\begin{aligned}
p(\mathbf{a}|\mathbf{e},\mathbf{f}) &= p(\mathbf{e},\mathbf{a}|\mathbf{f})/p(\mathbf{e}|\mathbf{f}) \\
&= \frac{\frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)})}{\frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} \sum_{i=0}^{l_f} t(e_j|f_i)} \\
&= \prod_{j=1}^{l_e} \frac{t(e_j|f_{a(j)})}{\sum_{i=0}^{l_f} t(e_j|f_i)}
\end{aligned}
$$

To collect counts of the English words being a translation of the Dutch words, we perform:

$$
c(e|f;\mathbf{e},\mathbf{f}) = \sum_a p(a|\mathbf{e},\mathbf{f}) \sum_{j=1}^{l_e} \delta(e,e_j)\delta(f,f_{a(j)})
$$

$$c(e|f;\mathbf{e},\mathbf{f}) = \frac{t(e|f)}{\sum_{i=0}^{l_f} t(e|f_i)} \sum_{j=1}^{l_e} \delta(e,e_j) \sum_{i=0}^{l_f} \delta(f,f_i)$$

The model can now be estimated as:

$$t(e|f;\mathbf{e},\mathbf{f}) = \frac{\sum_{(\mathbf{e},\mathbf{f})} c(e|f;\mathbf{e},\mathbf{f}))}{\sum_e \sum_{(\mathbf{e},\mathbf{f})} c(e|f;\mathbf{e},\mathbf{f}))}$$

The pseudocode for IBM Model 1 and the EM algorithm

```
Input: set of sentence pairs $(\mathbf{e},\mathbf{f})$
Output: translation prob. $t(e|f)$
 1: initialize $t(e|f)$ uniformly
 2: while not converged do
 3:    // initialize
 4:    count$(e|f) = 0$ for all $e, f$
 5:    total$(f) = 0$ for all $f$
 6:    for all sentence pairs $(\mathbf{e},\mathbf{f})$ do
 7:       // compute normalization
 8:       for all words $e$ in $\mathbf{e}$ do
 9:          s-total$(e) = 0$
10:          for all words $f$ in $\mathbf{f}$ do
11:             s-total$(e)$ += $t(e|f)$
12:          end for
13:       end for
14:       // collect counts
15:       for all words $e$ in $\mathbf{e}$ do
16:          for all words $f$ in $\mathbf{f}$ do
17:             count$(e|f)$ += $\frac{t(e|f)}{\text{s-total}(e)}$
18:             total$(f)$ += $\frac{t(e|f)}{\text{s-total}(e)}$
19:          end for
20:       end for
21:    end for
22:    // estimate probabilities
23:    for all foreign words $f$ do
24:       for all English words $e$ do
25:          $t(e|f) = \frac{\text{count}(e|f)}{\text{total}(f)}$
26:       end for
27:    end for
28: end while
```

Note: The above images use ‘f’ instead of ‘d’.

## MODULES IMPLEMENTED

As mentioned above, we have created different modules for implementing our project. They are Main.py, TrainModels.py, TestModels.py, IBMModel1.py, EnglishtoDutch.py, DutchtoEnglish.py, CosineSimilarity.py, Common.py and Jaccard.py.
Details related to each module are described below.

### 1.Main.py

This file is the starting point of our project. It contains the control code for the project and execution of various modules. This is the file that we will run using the command 'python main.py' to start the trainer/translator. It contains code that creates a simple command-line interface and various options available in program.

### 2.ModelTrainer.py

This module trains our IR model. The functions declared in this module are described below:

- create_tokenized_sentences(content_list, max_index): This function takes 2 inputs max_index (denotes maximum number of words that can exist in a list) and content_list (denotes a sentence ). It returns a list of words present in the sentence.
- train_models(): This function is used to train our model. The model trained

### 3.IBMModel1.py

This module is the implementation of the IBM model 1 for language translation. The functions declared in this module are described below:

- expectation_maximization(dutch_word_dict, english_word_dict, dutch_sentences, english_sentences): This function populates e/f matrix as described in IBM Model 1.
- get_translation_prob(e,f,t,e_dict,f_dict): It returns the probability of sentence e being translated to sentence f.

### 4.EnglishtoDutch.py

This module implements the translation of sentences from English to Dutch in our project. The functions declared in this module are described below:

- create_tokenized_sentences(content_list): This function takes an input sentence and returns a list of words contained in the sentence.
- translate(): This function translates given query English sentence into a Dutch sentence. It takes input from a file named 'input.txt' and the result of translation gets stored in a file 'output.txt' (Here we assume the files input.txt and output.txt are present in the directory).

### 5.DutchtoEnglish.py

This module implements the translation of sentences from Dutch to English in our project. The functions declared in this module are described below:

- create_tokenized_sentences(content_list): This function takes an input sentence and returns a list of words contained in the sentence.
- translate(): This function translates given query Dutch sentence into an English sentence. The result of translation gets stored in a file ‘output.txt’.

6.<u>CosineSimilarity.py</u>

This module calculates the cosine similarity between two sentences. The functions declared in this module are described below:

- calcCosineSim(): This function calculates the cosine similarity between the translated sentence by our model and actual translation. The translated sentence is taken from a file ‘output.txt’ and the actual sentence from ‘actual.txt’ and the similarity score is displayed on the screen (Here we assume that a file actual.txt is present in the directory and contains correct translation).

7.<u>Jaccard.py</u>

This module calculates the Jaccard Coefficient between two sentences. The functions declared in this module are described below:

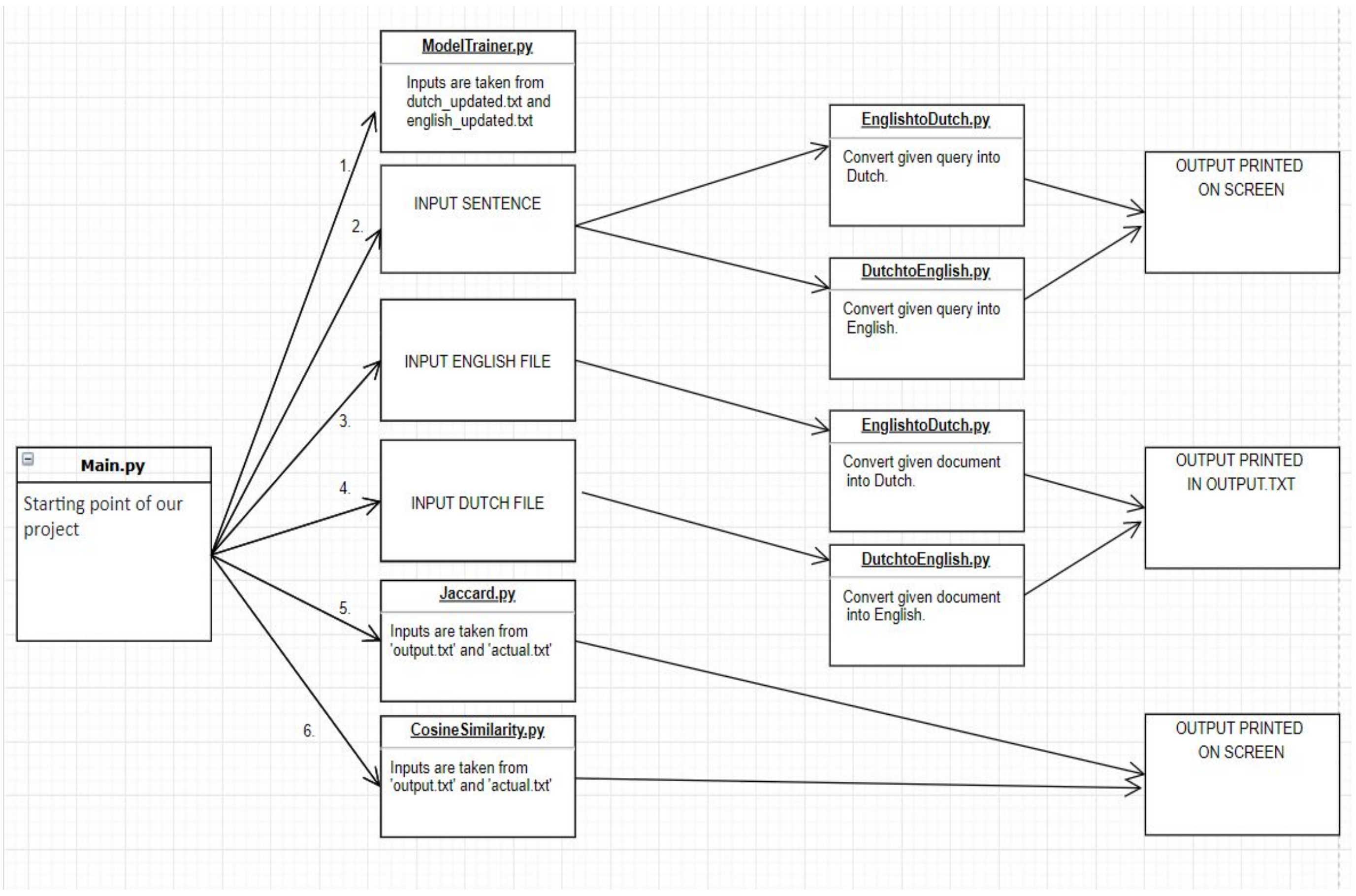
- calcJaccardCoeff(): This function calculates the Jaccard Coefficient between the translated sentence by our model and actual translation. The translated sentence is taken from a file ‘output.txt’ and the actual sentence from ‘actual.txt’ and the similarity score is displayed on the screen(Here we assume that a file actual.txt is present in the directory and contains correct translation).

8.<u>Utils.py</u>

This file contains the commonly used utility functions throughout our project. The functions declared in this module are described below:

- is_converged(new, old, num_of_iterations): This function checks whether the probabilities in e/f matrix converges. It returns TRUE/FALSE according to result.
- nCr(n,r): This function calculates mathematical expression nCr. It takes two integers n and r as input and returns nCr value.

## CONTROL FLOW OF THE PROGRAM



Main.py
Starting point of our project
1.
ModelTrainer.py
Inputs are taken from dutch_updated.txt and english_updated.txt
2.
INPUT SENTENCE
EnglishtoDutch.py
Convert given query into Dutch.
DutchtoEnglish.py
Convert given query into English.
OUTPUT PRINTED ON SCREEN
3.
INPUT ENGLISH FILE
4.
INPUT DUTCH FILE
EnglishtoDutch.py
Convert given document into Dutch.
DutchtoEnglish.py
Convert given document into English.
OUTPUT PRINTED IN OUTPUT.TXT
5.
Jaccard.py
Inputs are taken from 'output.txt' and 'actual.txt'
6.
CosineSimilarity.py
Inputs are taken from 'output.txt' and 'actual.txt'
OUTPUT PRINTED ON SCREEN