A DESIGN REPORT

ON

CROSS LANGUAGE INFORMATION RETRIEVAL

BY

Akshit Khanna	2017A7PS0023P	B.E. (Hons.) Computer Science
Vitthal Bhandari	2017A7PS0136P	B.E. (Hons.) Computer Science
Prakalp Tiwari	2017A7PS0137P	B.E. (Hons.) Computer Science
Ayush Garg	2017A7PS0193P	B.E. (Hons.) Computer Science
Aayush Nagpal	2017A7PS0219P	B.E. (Hons.) Computer Science

Prepared in partial fulfilment of the course

INFORMATION RETRIEVAL - CS F469

SUBMITTED TO

Dr. Lavika Goel

Assistant Professor

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

1. INTRODUCTION

Cross-language information retrieval (CLIR) is a subfield of information retrieval dealing with retrieving information written in a language different from the language of the user's query.

The IBM Models are a sequence of models with increasing complexity, starting with lexical translation probabilities, adding models for reordering and word duplication.

In IBM models, a sentence aligned bilingual corpora is used. However, there is no alignment of words. In IBM Model 1, the only consideration is lexical translations of words which means that we are trying to find word translation probabilities. By using an expectation-maximization algorithm for solving the two-sided problem, which if we know alignment of words, we can investigate word translation probabilities and if we know word translation probabilities, we can investigate alignment probabilities.

Because IBM Model 1 does not consider the alignment of the words and takes all alignment probabilities equally, a new model was introduced by IBM which is IBM Model 2. In IBM Model 2, in addition to the lexical translation model, alignment probabilities were introduced for probability calculation. The result of IBM Model 2 outputs alignment probabilities and word translation probabilities.

In our assignment, we have implemented the code for IBM models 1 and 2 to create a translator and we have calculated our accuracy based on cosine similarity and jaccard coefficient.

Since we have implemented the entire code in Jupyter notebook, we have provided the link of our entire code below :

https://drive.google.com/drive/folders/1hZFwQDIdtTxtQhI9aJvRvWSyCH5sHaOY?usp=sharing

2. PROGRAM INTERFACE

The program has been implemented in Jupyter Notebook. The entire code is written in python 3.7.1..

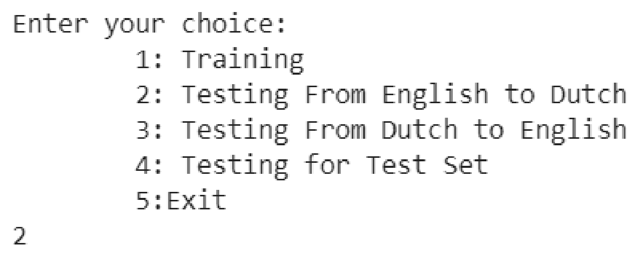
Once the user runs the code, he/she is prompted to enter a choice between 1 to 5. Choice 1 allows the user to train the data, choice 2 allows for testing From English to Dutch, choice 3 allows for testing From Dutch to English and choice 4 allows for testing on unknown dataset. Additionally, by choosing option 5, user can exit the program.

Upon the selection of either option 1, the user can choose a model of his/her choice - only IBM model 1 or both-IBM model 1 and IBM model 2.

Similarly upon the selection of either option 2 or 3, the user is allowed test either training or validation dataset. Post that, the user can choose a model of his/her choice - only IBM model 1 or both- IBM model 1 and IBM model 2.

3. PROGRAM EXECUTION

After execution, user will be asked to choose what he/she wants to do. The screenshot for the options is as follows:

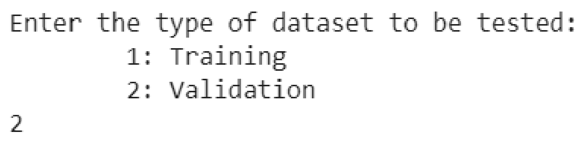


```
Enter your choice:
        1: Training
        2: Testing From English to Dutch
        3: Testing From Dutch to English
        4: Testing for Test Set
        5:Exit
2
```

Here there are five options, if the user chooses option 5, the program terminates.

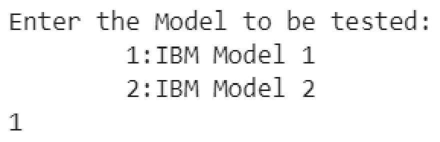
If the user chooses training(option 1), program starts to retrain itself, however, this operation can take time ranging from minutes to hours depending upon the size of the training dataset. So one should perform this operation as less frequently as possible.

If user chooses either option 2 or option 3 then the testing phase starts and the following options come(here the user has chosen option 3):



```
Enter the type of dataset to be tested:
        1: Training
        2: Validation
2
```

Here user chooses which kind of dataset to test. In this example the user chooses validation dataset. Upon choosing option 2(say), the user is asked to choose the IBM model of his/her choice as follows:



```
Enter the Model to be tested:
        1:IBM Model 1
        2:IBM Model 2
1
```

Once the user chooses his/her choice of model, the corresponding output consisting of cosine similarity, jaccard coefficient for each testing sentence pair and average cosine similarity, jaccard coefficient for all the test cases are displayed.

Option provides user with the ability to test on unknown dataset.

4. PROGRAM STRUCTURE

For the program, Python programming language is used.

4.1. Program modules

A number of interrelated methods have been implemented in our code. Their functionalities have been explained below in order of their occurrence in the code while the corresponding code blocks have been included in the appendix.

4.1.1. Header files : A number of header files have been used in the code. Except for the nltk library, which has been used for preprocessing the sentences to tokenize them, no other external library has been used for implementing the models.

4.1.2. converge_limit() : This method is used for running the whole EM-algorithm for a fixed number of iterations(usually 10). The number of iterations is passed as an argument to this method.

4.1.3. ibm_model_1() : This is the Expectation Maximization step required in IBM Model 1. EM Algorithm consists of two steps:

1. Expectation-Step: Apply model to the data using the model, assign probabilities to possible values

2. Maximization-Step: Estimate model from data

   (a) take assign values as fact

   (b) collect counts (weighted by probabilities)

   (c) estimate model from counts

Additionally, we have computed the probabilities with and without laplace smoothing. It has been observed that inclusion of laplace smoothing drastically improves the translational probabilities.

4.1.4. ibm_model_2() : The IBM Model 2 has an additional model for alignment that is not present in Model 1.The IBM Model 2 addressed this issue by modeling the translation of a foreign input word in position i to a native language word in position j using an alignment probability distribution defined as:

$$a(i \vee j, l_e, l_f)$$

In the above equation, the length of the input sentence f is denoted as l_f, and the length of the translated sentence e as l_e.

We have also incorporated laplace smoothing as a way to improve accuracy of IBM model 2.

4.1.5. sentences_tokenized() : This method does all of the pre-processing of the dataset. First it converts all the sentences into lowercase. Then it removes all punctuation marks in the dataset.

Finally, it tokenizes the processed sentences into a token stream and populates the translational dictionary .

4.1.6. train_models() : This method trains our model by taking a portion of the given dataset(say 10000 sentences).

4.1.7 get_tokens_of_sentence() : This method retrieves the generated token of each sentence.

4.1.8 testing_sentence() : This method returns the final translated sentence.

4.1.9 vector_similarity() : This method calculates the cosine similarity and jaccard coefficient between any two sentences using the formulas given below:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

Where Ai and Bi are components of vectors A and B respectively.

Similarly Jaccard coefficient is calculated as follows:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

4.1.10 test_model() : The method below is used for testing our model on training and validation dataset. It takes into account the direction of translation as specified by the user(dutch to english or english to dutch).

5. INNOVATION

As part of adding innovation to our assignment, we have implemented several new features in our code such as:

1. We implemented **IBM model 2** even though the pseudocodes were not available to us as part of our course structure. This way we were able to improve the efficiency of our translator.
2. We added **Laplace smoothing** in the codes for IBM model 1 and 2 to improve the accuracy of our translator. The effect was substantial.
3. To avoid computational difficulties we limited the size of each sentence to 10 words.

6. IMPROVEMENTS AND EXTENSIONS

The dataset used for training had approximately 2 million aligned sentences, however, since the computing power available to us was not enough to use all of it, we used various subsets of it which had 10000-50000 sentences.

Additionally, for computation time issues, we used sentences with length at most 10. So, these choices decreased the size of the dictionary and caused the translation of some words not to be found in the dictionary.

If the size of the corpus increases, the accuracy of the translator will increase.

7. DIFFICULTIES ENCOUNTERED

There are some difficulties encountered when developing the system. Especially for Model 2 since the pseudo code was not in the book as well as in the course slides.

So we had to search the pseudocode on internet. Also, some points in the pseudocode were not clear like calculating *null* value. In addition to these difficulties, there were also some computational problems. For instance, the sentences longer than 10 words caused IBM Model 2 to work for a longer duration of time. So, we limited the sentence length with 10 words.

Moreover, since iterations take considerable amount of time, we limited the iteration number to 10. However, the models provided by us are not perfect and there is a scope for improvement.

8. CONCLUSION

This project has been a learning experience for statistical machine translation. We learned the general logic of statistical machine translation and have a better idea about IBM Models now. The logic derived and its causes also taught us how to think about machine translation problems. It excited me to produce our own machine translation system for other languages.