A. **Specific problem definition** :

In this homework, we have to address the problem of coreference resolution. The data given in this homework is GAP, a human-labeled corpus of ambiguous pronoun-name pairs derived from Wikipedia. There are 4 steps to do this homework: (1) download Gap coreference repository from the given link in guide file (2) develop program that resolves ambiguous pronouns that refer to names (3) access the performance of the program in step (2) against 4000 pairs in gap-test.tsv evaluated by Precision, Recall, F-score and Bias ratio with two task settings: snippet-context and page-context.

B. Discuss the result:

a. Idea for the code to prove the reasonability of my result:

The only issue in this homework is : from a given string of many sentences with one annotated pronoun P and two annotated people name A, B, how we can determine whether P,A are coreferential and whether P,B are coreferential. My idea for this issue is : from the data read from file gap-development.tsv , we will create a training set and create a classifier by nltk.MaxentClassifier to train this training set, after that, with each line U in file gap-test.tsv, we will use the classifier to determine whether each pair (P,A) and (P,B) of U is coreferential. We have 2 tasks: snippet-context and page-context. With the snippet-context, to create the training set, I initiate an empty list named train_data, then write program to read each line in file gap-development.tsv , after that, with each line X, we will read 8 information: ID, text, P (the pronoun) , A (the first name) , A-coref (the boolean value corresponding to whether (P,A) is coreferential), B (the second name) , B-coref (the boolean value corresponding to whether (P,B) is coreferential), URL(the link to the article on Wikipedia). From these information, we will generate a dictionary of features for pair (P,A) as follow : initiate an empty dictionary dict_ = {} , the first feature is the couple of A and P, that means dict_[“noun_pronoun”] = (A, P) . The second feature is the distance of A and P on the test, that means the number of words between A and P on the test. The idea behind this feature is that couple (name,pronoun) with shorter distance would be more likely to be coreferential. To have the next feature, we need to write program for chunking a sentence, to do this, I write a chunk grammar and then use nltk. RegexpParser to create a chunk parser. Then, we can use this parser for chunking sentences of the text (text is a data that I have defined above) , and the result is chunk trees of sentences of the text. Now, we can determine the position of A and P on these trees, we assume that the 2 subtrees containing A and P are t1 and t2, then the next feature to add to the dictionary dict_ is the distance of A and P on these trees of chunks, that means the number of subtrees between t1 and t2. The next feature is to check whether A is a subject or an object in its sentence. I choose this feature because according to some linguistic researches about coreference that I found on the internet, entities in the subject position is more likely to be referred to than entities in the object position. Assuming that A is contained in sentence M, then to check whether A is subject of M or not, we can use a very simple idea: firstly, chunking the sentence M to have a tree T, then among subtrees of T, if we can find a NP phrase (noun phrase) that is the child of T and the sibling of at least one VP phrase (verb phrase), then this NP phrase has high probability to be the subject of sentence M. Therefore, we can check, if A is contained in a NP subtree that is the child of T and this NP subtree has at least one VP sibling subtree, then dict_[“check_subject”]= True, otherwise, dict_[“check_subject”] = False, and this is our fourth feature. The final feature is based on a simple idea: more times A appears in the text (the text data is defined above) , then more likely that A is coreferential to the pronoun P. Therefore, we count the number of occurrences of A inside the text, and this is our final feature. Then, we add the couple (dict_ , A-coref) to the list train_data. Similarly, we create a dictionary of features for (B,P) in the same way, then let name it second_dict, and we add the couple (second_dict, B-coref) to train_data. By doing this for all lines of the file gap-development.tsv, we have created the train set to be used for training for the task

snippet-context. With the remaining task, page-context, we will generate the training set in the same way, but we will add one more feature when we create the dictionary of feature. In more detail, with the line X that we have read from file gap-development.tsv, we get its URL and then we can use this link to get the title of its Wikipedia article, let assume that this title is the string Y. Then, we have 2 strings A and Y, we will check whether string A and Y have common substring, if they have, then dict_[“overlap_title] = True, otherwise, dict_[“overlap_title] = False and this is the feature that we have append for task page-context. After generating training data, we use nltk.MaxentClassifier to create a classifier, and then we use the training data above to train this classifier. Then, the next step is to predict the coreference relations of each line in file gap-test.tsv. To do this, when we read each line Z of this file, with pair (P,A) of Z where P is pronoun and A is name, we will create 2 dictionaries of features of (P,A) in 2 manners snippet-context and page-context in the way described above, then we use these dictionaries as the input for the classifier that we have trained, and then the output of the classifier would be True or False. This is the prediction of our model about whether P,A is coreferential. We do in the same way for pair (P,B) of line Z. By doing this, we can predict the coreference relations of each line in file gap-test.tsv

b. Evaluating result:

After running the code to create 2 tsv files for snippet-context and page-context tasks, I run the gap-scorer file and the result look quite good: for snippet-context, overall recall = 48.9 , precision = 57.0 , f1= 52.6 and Bias = 0.93 , for page-context, there is an improvement in all scores : overall recall = 49.1, precision = 57.2 , f1= 52.8 and Bias = 0.92. In case of page-context, we use URL to add one more feature, therefore, it is reasonable to see an improvement in scores for page-context task. The bias scores for both tasks are high : 0.93 and 0.92, thus it indicates that my program eliminates the gender bias very well. However, I think that the recall score is not very high and still need more improvement in the future.

c. For improvement: As I mentioned above, when generating the training set, one of features that I choose is the distance between names A,B and the pronoun P on the trees of chunks. However, this features involve the task of chunking a sentence and in fact, to do this, I write a chunk grammar and then use nltk to create a parser to parse the sentence. But after many experiments, I realize that this parser cannot give a perfect chunking for every sentence. Therefore, for improvement in the future, I believe that if we use an external model for chunking sentences, then we can have a better chunking performance, and then give more accurate feature for training and as a result, this helps improve the overall result. Secondly, there is one more feature that we can add to the dictionary of features , that is : with the name A and the pronoun P, we will determine whether A is a subject or an object in its sentence, and we also determine whether P is a subject or an object in its sentence, then, if A and P are both subjects or both objects, then we return True for this feature, otherwise, if A is subject while P is object or A is object while P is subject, then we return False. The idea behind this feature is very simple: if A and P are both subjects or objects at the same time, then they are more likely to be coreferential to each other. But in fact, I do not add this feature to my code implementation because to determine subjects and objects of a sentence with high accuracy, we need to generate a dependency tree for that sentence, but as far as I know, dependency tree requires external model. Therefore, I decided not to add this feature, but for the purpose of improving performance of this code implementation in the future, I believe that this feature is a good one to add. Finally, in this implementation, to create classifier for training, we have to use available models in nltk (for example nltk.MaxentClassifier), but I think that these models are not best ones for training and classifying. For the purpose of improvement, I believe that using external models , for example sklearn will give a better performance and can improve our overall result.