A. Specific problem definition:

In this homework, the task is to search MEDLINE abstract to collect 100 sentences, each of which contains at least one of five verbs activate, inhibit , bind, A and B ,that A is a verb with positive action and B is a verb with negative action (step 1). Our next task is to provide annotations to these sentences for triples <X,ACTION,Y> to extract (step 2). The next task is to choose 80 sentences and develop a relation extraction model based on these 80 sentences, and after that, we have to access its performance (step 3). The final task is to apply the module of previous step to the remaining 20 sentences and access its performance. (step 4)

B. Discuss the result:

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

The main issue of this homework is the step 3, that after choosing 80 sentences with annotated triples, we have to develop a relation extraction model based on these sentences. My solution for this issue is: Firstly, I find way for chunking each sentence into a tree of chunks. To do this, I create a classifier-based chunker and train it by the corpus conll2000 in nltk. In fact, this classifier-based chunker is introduced in chapter 7 of the textbook, and it is based just on nltk, therefore, this chunker is not an external model, however, I use it because it seems to work quite well. After training, now I have a chunker to use for next steps. In this homework, I choose 5 verbs and form a list: V= [activate, inhibit, bind, prevent, accelerate].

Secondly, we will create dataset for training by using 80 sentences that we have chosen. The first step is to create an empty list named train_data. The idea is that, from a sentence, we will collect all possible triples, and then with each triple M, we will extract some of its features to create a dictionary named dict_fea, and finally, by looking at the list of correct triples that we have manually annotated for that sentence, we will know whether triple M is a correct triple of the form <X,ACTION,Y> or not. If triple M is a correct triple, then we append the couple (dict_fea, 1) to the list train_data, else we append the couple (dict_fea, 0) to the list_train_data (1 is the label for correct triple and 0 is the label for incorrect triple) . Thus, we have 2 main problems: firstly, from a sentence, we have to find all possible triples from it, and secondly, with each triple, we have to extract good features of that triple for training. To solve the first problem, with each sentence S among these 80 sentences, we will use the chunker to create a tree of chunks for that sentence. Next, we determine all occurrence of each of 5 verbs in list V in the sentence S. Assume that we have V_1,V_2,… V_n appearing in S that each V_1 , V_2 , … V_n is a verb in the list V=[activate, inhibit, bind, prevent, accelerate] or reflected form of that verb. We then create an empty list named list_triples to collect all possible triples. In previous step, we have used the chunker to create a tree of chunks for sentence S, assume the name of this tree is T, then with each verb V_i defined above, we use the tree T to determine noun phrases surrounding V_i in sentence S. Assume noun_phrase1, noun_phrase2 are 2 noun phrases that appear before V_i in sentence S and they are closest to V_i. Similarly, assuming noun_phrase3, noun_phrase4 are 2 noun phrases that appear after V_i in sentence S and they are closet to V_i . Then we have 4 possible triples (noun_phrase1, V_i , noun_phrase3), (noun_phrase1, V_i, noun_phrase4), (noun_phrase2, V_i, noun_phrase3), (noun_phrase2, V_i , noun_phrase4). By doing this for all verb V_1,V_2, … V_n , we can create a list of possible triples for sentence S, and we append all of these triples to list_triples. Next, from each triple (A,B,C) from list_triples, we have to extract its feature. In the code implementation, I extract some of features of (A,B,C) that I think these features are reasonable. For example, we know that A and C are noun phrases, and I choose the first feature to be the distance between A and C in the tree of chunks of S. I choose the second feature to be a true/false variable, it would be True if there is no noun phrase between A and C in the tree of chunks and it would be False if there is at least one. There are some other features, that more detail is given in the comment of code in file .py . From now on, we have collected all possible triples and create a dictionary of features for each triple. With each triple, we append the couple (dictionary of features of triple, label of triple) to the list train_set and now, we have the train dataset. I create “classifier = nltk.MaxentClassifier” to use for training. After this training step, with a new sentence K, I would give all predicted triples <X,ACTION,Y> annotations to it as following: firstly, creating an empty list name predicted_triples. Next, collecting all possible triples from K and create a dictionary of features for each triple by doing in the same way as the above part. Then, with each triple, we use its dictionary of features as the input for the classifier that we have trained, and this classifier would output the label 0 or 1. If the output label is 1, then we would append this triple to the list predicted_triples. By doing this, we collect all triples predicted by our module for a sentence. To access the performance of 80 train sentences and 20 test sentences, we use the triples annotations that we have manually annotated for each sentence and use the formula: precision = (number of true prediction)/(number of predicted triples), recall = (number of true predictions)/(number of true relation) and F1score is the harmonic mean of precision and recall.

b. **Evaluating result**: The result seems to be good, the precision, recall, f1score of 80 train sentences are 0.608333, 0.9125, 0.7299 and those for 20 test sentences are 0.6333, 0.95, 0.7599. In my opinion, these scores are high enough and there is no much difference between scores for train sentences and test sentences, thus it implies that our module does not get overfitting issue and this result is very reasonable. However, we can observe that the precision scores are still not very high, thus, our module still need more improvement.

c. **For improvement**: In this code, I train a classifier-based chunker to use for chunking for next steps, and in fact, this chunker works quite well but not absolutely good because it still give wrong tags for some chunks of some sentences, so I believe that an external model for chunking would work better and can improve the result. Secondly, the problem of training is that the data is not much (just 80 sentences) and therefore, if we want to improve the result, we need to collect more sentences that contain at least one relation from MEDLINE. Thirdly, as I mentioned in part (a), in the step of collecting all possible triples from a sentence S, with every occurrence V_i of one of five verbs [activate, inhibit, bind, prevent, accelerate] or their reflected forms, I would find 2 closest noun phrases N1,N2 appearing before occurrence of V_i in S , and find 2 closest noun phrases N3,N4 appearing after the occurrence of V_i in S , and we have 4 possible triples (N1,V_i , N3), (N1, V_i , N4), (N2, V_i ,N3), (N2, V_i , N4). However, the problem is that among these 4 triples, there is at most one correct relation, thus, the portion of triples with label 1 in the train_set is at most 25% and this leads to the imbalance in the data with label 0 and 1 for training. So, to improve the quality of our result, we need to balance the train_set by using PUBMED to collect more sentences containing at least one relation, and then extracting correct triples (triples that show the relation <X,ACTION,Y>) from these sentences, and then appending them to train_set to increase the number of data with label 1.