COMPGI19 Assignment2 – Event Extraction

Weijie Huang (15042248)
weijie.huang@ucl.ac.uk
Shuo Wang (15033967)
ucabssw@ucl.ac.uk
Dongdong Zhang (15021014)
dongdong.zhang.15@ucl.ac.uk
MSc Web Science and Big Data Analytics

Abstract

In this task, it involves some problem about biological information processes, which are from the BioNLP Shared Task on "biomedical event extraction". That contains three aspects of knowledge: generative VS discriminative training and modeling, feature engineering, structured prediction. This report will describe the various problems of this assignment. Firstly, the perceptron and average perceptron algorithm are mentioned. Then, the different feature functions will be discussed. After that, a joint perceptron method that combines trigger and argument extraction. Finally, it analyses some errors on predicting processes and extends a conditional likelihood as jointed part.

1 Problem 1: Implementation of Perceptron Algorithm

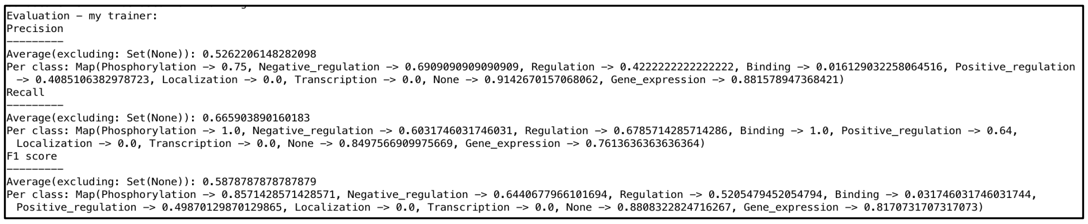
In this problem, a perceptron algorithm is applied to a log-linear model for Trigger extraction, which is used to predict the trigger words' labels. The log-linear model has some binary feature functions $\phi_i(\boldsymbol{x}, c)$ including the input(x) and their label(c), and output is one if input matches feature, otherwise it is 0. The conditional probability is defined as a log-linear, represented as follow:

$$P_{\boldsymbol{\lambda}}(c|\boldsymbol{x}) = \frac{1}{Z_{\boldsymbol{x}}} \exp\left[\sum_i \phi_i(\boldsymbol{x}, c)\lambda_i\right] = \frac{1}{Z_{\boldsymbol{x}}} \exp < \boldsymbol{\phi}(\boldsymbol{x}, c), \boldsymbol{\lambda} >$$

This equation is the inner production of the binary vector $\boldsymbol{\phi}(\boldsymbol{x}, c)$ with a parameter $\boldsymbol{\lambda}$, and then it is divided by $Z_{\boldsymbol{x}}$ to get the normalization expression. To obtain the best prediction, $\boldsymbol{\lambda}$ is trained by perceptron algorithm. Firstly, from the initial codes, it states the learning rate is 1, the repeated time is 2 and the initial $\boldsymbol{\lambda} = [0, 0 \ldots, 0]$. The iteration is achieved by a loop, and a nesting loop is added to traverse the whole instances. Furthermore, for each case, it performs precompiled method "predict" to find the label of the maximum inner product value from each label $c' = \arg\max_c P_{\boldsymbol{\lambda}}(c|\boldsymbol{x})$. If the predicted label is not equal to the appropriate label $c' \neq c_i$, the weight parameter should be updated using:

$$\boldsymbol{\lambda} = \boldsymbol{\lambda} + \boldsymbol{\phi}(\boldsymbol{x}_i, c_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, c')$$

Figure 1 shows that the results of the built algorithm, which are same precisely as the pre-complied with the default settings and call the default feature functions.

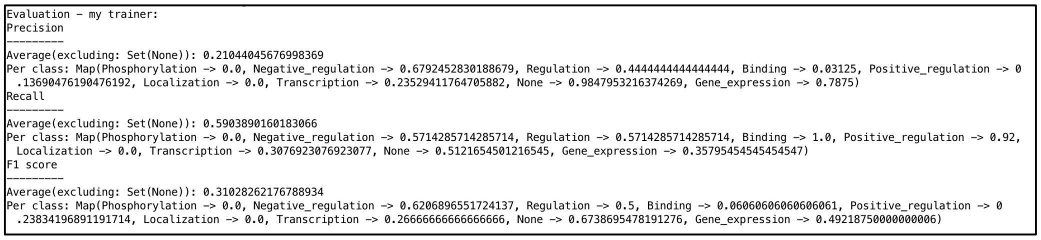


```
Evaluation - my trainer:
Precision
---------
Average(excluding: Set(None)): 0.5262206148282098
Per class: Map(Phosphorylation -> 0.75, Negative_regulation -> 0.6909090909090909, Regulation -> 0.4222222222222222, Binding -> 0.016129032258064516, Positive_regulation
 -> 0.4085106382978723, Localization -> 0.0, Transcription -> 0.0, None -> 0.9142670157068062, Gene_expression -> 0.881578947368421)
Recall
---------
Average(excluding: Set(None)): 0.665903890160183
Per class: Map(Phosphorylation -> 1.0, Negative_regulation -> 0.6031746031746031, Regulation -> 0.6785714285714286, Binding -> 1.0, Positive_regulation -> 0.64,
 Localization -> 0.0, Transcription -> 0.0, None -> 0.8497566909975669, Gene_expression -> 0.7613636363636364)
F1 score
---------
Average(excluding: Set(None)): 0.5878787878787879
Per class: Map(Phosphorylation -> 0.8571428571428571, Negative_regulation -> 0.6440677966101694, Regulation -> 0.5205479452054794, Binding -> 0.031746031746031744,
 Positive_regulation -> 0.49870129870129865, Localization -> 0.0, Transcription -> 0.0, None -> 0.8808322824716267, Gene_expression -> 0.8170731707317073)
```

Figure 1: The completed trainer results of perceptron algorithm

2 Problem 2: Implementation of Average Perceptron

Comparing with Problem 1, another vector is added to record the total weight's value when each update happens. Also, this vector is divided by the time the weight being updated before return. Figure 2 states that the results are same with the pre-compiled trainer completely.



```
Evaluation - my trainer:
Precision
---------
Average(excluding: Set(None)): 0.21044045676998369
Per class: Map(Phosphorylation -> 0.0, Negative_regulation -> 0.6792452830188679, Regulation -> 0.4444444444444444, Binding -> 0.03125, Positive_regulation -> 0
 .13690476190476192, Localization -> 0.0, Transcription -> 0.23529411764705882, None -> 0.9847953216374269, Gene_expression -> 0.7875)
Recall
---------
Average(excluding: Set(None)): 0.5903890160183066
Per class: Map(Phosphorylation -> 0.0, Negative_regulation -> 0.5714285714285714, Regulation -> 0.5714285714285714, Binding -> 1.0, Positive_regulation -> 0.92,
 Localization -> 0.0, Transcription -> 0.3076923076923077, None -> 0.5121654501216545, Gene_expression -> 0.35795454545454547)
F1 score
---------
Average(excluding: Set(None)): 0.31028262176788934
Per class: Map(Phosphorylation -> 0.0, Negative_regulation -> 0.6206896551724137, Regulation -> 0.5, Binding -> 0.06060606060606061, Positive_regulation -> 0
 .23834196891191714, Localization -> 0.0, Transcription -> 0.26666666666666666, None -> 0.6738695478191276, Gene_expression -> 0.49218750000000006)
```

Figure 2: The completed trainer results of average perceptron algorithm

The implementation of the average perceptron is slow because there are some unnecessary updates on accumulation. If the weight is not updated, it can be ignored on adding process. Moreover, in the normal method, each element in *returnWeights* will be divided by a constant that is time-consuming. Thus, there are two methods to solve that separately to speed up. The first way is judging the element of the weight on each update step. If the element is not zero, it will be added to the corresponding element in *returnWeights*; otherwise, this element will be passed. Therefore, it does not need to traverse the whole map to add zero. Furthermore, the sum of the value of weight replaces the averaged one as a return. At the same time, it will not impact the average algorithm, because the denominator is a constant and the ratio of every feature's weight does not change. For example, if the data size is changed from 100 to 500, the time is speeded 25% after using these two methods.

3 Problem 3: Feature Engineering and Evaluation

This section will separate into two parts, trigger classifier and argument classifier, and each one will contain feature description, examples and the best set of two algorithms. In perceptron algorithm, iteration is set to 12.

3.1 Trigger classifier

3.1.1 Trigger Feature Description & Examples

There are eleven features for trigger we used, but pick four different types of feature to describe precisely.

1. Phosphorylation & … & Protein_catabolism & Localization Feature (Lexical Feature)
   Feature: $\Phi_{0,c'}(x, c) = 1, \text{if word}_x = w' \wedge c = c'$
   According to the rule observed in train set, it is more likely for "Phosphorylated" to be a Phosphorylation, "bind" to be Binding. It means, for each type of label, there is a set of words that has a high probability to be labelled as this type.
   Example: There are five records about "Phosphorylated" in train set, and all of them are labelled as Phosphorylation.
2. Nearby Proteins (Entity-based Feature)
   Feature: $\Phi_{1,c'}(x, c) = 1, \text{if nearby words of x contains(protein)} \wedge \text{protein.label} = p' \wedge c = c'$
   It is more likely for a trigger to find a neighboring protein to build an event. Thus, this feature is to find whether there is any protein close to this candidate and the type of this protein.
   Example: In train set, there are five records that near the protein "bcr-abl", and all of them are labelled as Gene_expression.
3. Dobj Feature (Syntax-based Feature)
   Feature: $\Phi_{2,c'}(x, c) = 1, if\ pos_x = "VB" \wedge\ dep(x, protein) = "Dobj"\ \wedge c = c'$
   According to the rule observed in train set, if the candidate is Verb and the relation between candidate and protein is dobj, then it is more likely to be labelled as Positive /

Negative Regulation.
Example: There are 134 records that the candidate is a verb, dependency's label with a protein is "Dobj", and then the probability of Negative_regulation and Positive_regulation is 0.209 and 0.336, larger than others.

4. Bigram and Trigram (another type)
Feature: $\Phi_{3,c'}(x, c) = 1, \text{if } \text{word}_{x-i} = w'_1 \wedge \text{word}_{x-i+1} = w'_2 \wedge c = c'$

$\Phi_{4,c'}(x, c) = 1, \text{if } \text{word}_{x-i} = w'_1 \wedge \text{word}_{x-i+1} = w'_2 \wedge \text{word}_{x-i+2} = w'_3 \wedge c = c'$

This method is taking the trigger word as a center and set a distance. A loop is built from $\text{candidate} - \text{distance}$ to $\text{candidate} + \text{distance}$, each time pick up two or three words as a feature. It is well known to find if there is any relation between triggers' label with adjacent words' collaboration. To gain the highest result, the distance of bigram is setting to four, and the trigram one is five.
Example: There are 14 records in train set that within the range, the ith word is "IRF-4", and the i+1th word is "expression", then the probability to be labelled as Regulation, Negative_regulation, Gene_expression is 0.36, 0.21, 0.43.

5. Trigger Bias
As provided, it means $c = c'$, used to adjust the probability of different trigger label.
Example: It is more likely to see label Negative_regulation, Positive_regulation, and Gene_expression, which has a high proportion. Thus, it is useful to adjust the whole distribution.

6. First Token of Trigger
It is used to find the relation between candidate itself, a relatively significant factor, and its label. Besides, the first trigger token has an excellent performance to represent the word, so the first token of the trigger is chosen.
Example: In the train set, if candidate equals to "bound", then its' label has approximately 100% chance to be Binding.

7. First Mention Word
Trigger words have closely linked to protein. Except the nested event, trigger always points to a protein to build an argument. According to the data structure of mention, mention can be a word or a phase so that the beginning token may be different with the end. However, first word has better performance to represent mention. Thus, the first word of mention is used.
Example: There are 241 records that the first mention word is "IRF-4", and the probability of None is 0.037, less than others.

8. Dependency (Syntax-based Feature)
Dependency was designed to represent the semantic relationships, and the type of relationships between two words if any grammatical relation exists.
Example: In the train set, there are ten records $\text{conj\_and}(\text{molecule} - 1, \text{molecule} - 1)$, and the probability of being labelled as Positive_regulation, Negative_ regulation, Gene_expression is 0.3, 0.2, 0.1 respectively, and the rest of records were labelled as other four types, different distribution with none dependency.

9. Words Before and After trigger
Like N-gram to represent the relationship with the particular context, this feature is used to take the context as a factor, precisely, words before and after the trigger candidate, which has a more significant influence on candidate's label.
Example: There are six records in train set that [before: to], [candidate: produce], [after: IL-10], and all of these are labelled as Gene_expression.

10. Dep Trigger (Syntax-based Feature)
Whether there is a direct link between two words can be taken to a useful feature, because trigger word always has a connection to another word.
Example: There are 1603 records in train set that involves in Dep dependency, and the probability of None is 0.13, less than others.

11. Mid Trigger
Whether the candidate located between two proteins is also a useful feature. Because trigger word always has to find the nearest protein to form an event.
Example: There are 912 records that the candidate locates between two proteins, and the probability of None is 0.07, less than others.

3.1.2 Best set of feature

Features are not independent but work together. It is useful to find the best set to make each feature's performance to the greatest extent. The best set of feature is the set with the highest F1 score. For perceptron and NB, it is different according to lots of trials. Results are shown below:

	Evaluation(average)%			Best Set of Feature (same index with the last section)
Learning algorithm	Precision	Recall	F1	
Perceptron	22.49	77.17	34.82	1.2.3.4.5.6.7.8.9.10.11
NB	15.42	79.37	25.82	5.6

Table1: Evaluation for trigger

For perceptron algorithm, according to experiments, the best set is all of these features mention above, with the final F1 score 34.82%. Regarding NB algorithm, the best set of features only contains two elements, Bias Feature and First Trigger Word. Because this method lacks training, the weight cannot change even if the predict is incorrect. It is apparent that based on this dataset and best feature set respectively, perceptron algorithm is more efficient to predict trigger.

3.2 Argument classifier

3.2.1 Argument Feature Description & Examples

In an event, a trigger may point to a protein or a nested event. Because the latter one is more complex, so arguments with the trigger to protein will be paid much more attention. There are nine features shown, but we pick four to describe precisely.

1. First Trigger word_Regulation_Cause (Lexical Feature)
Feature: $\Phi'_{0,c'}(\mathrm{x,c}) = 1, \mathrm{if\ Regula}tionList\ contains(word_{x.trigger}) \wedge c = c'$
According to the rule observed in train set, if the trigger word is high likely to be labelled as Regulation, such like increasing, accumulation, enhanced, increased, upregulated, etc. are include in a list called RegulationList, and then the argument is more likely to be labelled as Cause. Because only Regulation and Positive/Negative regulation can have Cause argument.
Example: There are 1011 records in train set that the trigger is one of the words in Regulation List, and the probability of Cause is 0.144, significant than others.

2. Is Protein & Trigger Word (Entity-based Feature)
Feature: $\Phi'_{1,c'}(\mathrm{x,c}) = 1, if\ x\ is\ protein\ \wedge\ word_{x.trigger} = w' \wedge c = c'$
Another important feature for argument is the trigger word and whether the candidate is a protein. Specifically, the argument always represents a trigger point to a protein, so if the trigger word is not *None*, and the argument candidate word is a protein, there would be high likely to form an argument.
Example: There are four records where a candidate is protein, and the trigger word is "deregulation". All of them are labelled as Theme.

3. Deps' label & IsProtein (Syntax-based Feature)
Feature: $\Phi'_{2,c'}(\mathrm{x,c}) = 1, \mathrm{if\ Dep's\ label} = \mathrm{l}' \wedge \mathrm{word_x is\ Protein} \wedge \mathrm{c} = \mathrm{c}'$
Like the feature Is Protein & Trigger Word mentioned above, the probability of an argument is related to the dependency and whether it is a protein.
Example: There are 848 records where dependency is "prep_of", and the candidate is a protein, and then the probability of Theme and Cause is 0.662 and 0.119, significant than others.

4. Trigram/Bigram_Argument
Feature: $\Phi_{3,c'}(x,c) = 1, if\ \ word_{x-i} = w'_1\ \wedge\ word_{x-i+1} = w'_2 \wedge c = c'$
$\Phi_{4,c'}(x,c) = 1, if\ \ word_{x-i} = w'_1\ \wedge\ word_{x-i+1} = w'_2 \wedge\ word_{x-i+2} = w'_3 \wedge c = c'$

Like the trigram/bigram feature in a trigger, this one is used to find if there is any relation between arguments' label with adjacent words' collaboration.
Example: There are 75 records where the trigram words are "gene" and "expression", then 48 of them are labelled as Theme, 24 of them are labelled as None, and the rest are Cause. The large difference with the distribution of the total number with these three labels, which means it may be related to trigram words slightly.

5. Label Bias_Argument
The first choice of feature to adjust the probability of every different label.
Example: It is more likely to see label Theme, which has a high proportion.
6. First Argument Word
Like the trigger feature *First Trigger Word*, the candidate itself is a significant factor.
Example: There are nine records in train set where a candidate is ":", and all of them are labelled as No*ne*.
7. Nearby Events' trigger word
In a single sentence, the trigger candidate has a link with nearby triggers. The same idea, for argument, maybe there is a relationship between argument candidates with nearby triggers. Thus, taking words before & after event candidate and the trigger candidate itself into account.
Example: There are 100 records where the word before a candidate is "of" and the word after candidate is "of", then the probability of Theme is 0.78.
8. Three Dependencies Count_Argument
The argument shows a relationship in an event, to some extent, the relationship is independent with grammatical structure.
Example: There are 354 records in train set that the dependency of a candidate with protein ("IRF-4"), and the probability of theme is 0.627, large than others.
9. posTag_Argument
The tag is a provided information, shows the characteristic of a word. It is a good feature to classify. For example, if a word is a noun, the probability of being an argument will be higher than a word that is not noun.
Example: There are 73 records where the candidate's tag is "VB", and 48 of them are None, 23 of them are Theme, the rest are Cause. It represents that if the word's tag is "VB", then it is more likely to be labelled as None.

3.2.2 Best set of feature

For perceptron algorithm, the final result F1 is 0.1253, and the best set contains all of these features. For NB algorithm, the best set only has three elements, Label Bias, First Argument Word and Is Protein & Trigger Word. Same reason with trigger feature, no iteration to optimize the weight if the predict is not correct, so other features is meaningless. According to the final result, perceptron algorithm is more efficient to predict argument.

	Evaluation(average)%			Best Set of Feature(same index with the last section)
Learning algorithm	Precision	Recall	F1	
Perceptron	5.92	86.18	11.07	1.2.3.4.5.6.7.8.9
NB	3.42	86.93	6.58	2.5.6

Table2: Evaluation for argument

Problem 4: Joint Perceptron

4.1 Unconstrained joint model

Firstly, the pseudo-code of the unconstrained joint model search routine is shown below:

```
1. algorithm Unconstrained joint model is
2.     input: a token $x$
3.         the set of trigger's labels $e_i, i = 1,2 \dots, n$
4.         the candidate argument tokens $c_j, j = 1, \dots, m$
```

```
5.                the set of arguments' labels  $a^i_{c_j}, i = 1,2 \dots, n$
6.                triggers' feature functions  $triggerFeature$  (vector)
7.                arguments' feature f $argumentFeature$  (vector)
8.                the current weight parameter  $\boldsymbol{\lambda}$
9.        output: the best trigger label  $e'$
10.               the best argument label  $\boldsymbol{a}' = (a'_{c1}, \dots, a'_{cm})$
11.       For each labels of trigger  $e_i, i = 1,2 \dots, n$
12.                                 $\mathrm{map}(e_i, score) \leftarrow \mathrm{map}(e_i, score) + (e_i \rightarrow$
                  $dot(triggerFeature(\boldsymbol{x}, e_i), \boldsymbol{\lambda}))$
13.       find  $map(e_{max}, \max score)$
14.       $e' \leftarrow e_{max}$
15.       For the candidate argument tokens  $c_j, j = 1, \dots, m$
16.               For each labels of arguments  $a^i_{c_j}, i = 1,2 \dots, n$
17.                                 $\mathrm{map}\left(a^i_{c_j}, score\right) \leftarrow \mathrm{map}\left(a^i_{c_j}, score\right) + a^i_{c_j} \rightarrow$
                        $dot(argumentFeature\left(\boldsymbol{x}, a^i_{c_j}\right), \boldsymbol{\lambda})$
18.               find  $map(a^{max}_{c_j}, \max score)$
19.               $a'_{c_j} \leftarrow a^{max}_{c_j}$
20.       $\boldsymbol{a}' \leftarrow (a'_{c1}, \dots, a'_{cm})$
21.       Return  $(e', \boldsymbol{a}')$
22.  end
```

In the beginning, the inputs contain a token input x, all the trigger and argument labels, the candidate arguments tokens that are from one event, the feature functions of trigger and argument as a vector that consists of input instance and label, and the current weight parameter. Then the outputs are the best labels that are obtained from the maximum scores tuple.

Secondly, for the trigger, all the label candidates are loaded to the "triggerFeature" vector by a loop and then it uses an inner product with the parameter vector, each candidate and corresponding scores recorded in a map structure. The next step is performing the maximum method of the map to seek the largest value of score and return the relevant trigger label as the best trigger label. Then, there is a nested loop structure to find the best labels of argument, because each input may have more than one arguments. Therefore, the first loop is traversal for all the candidate argument tokens. After that, the same method with trigger section is applied to achieve the best label, but in this part, it will produce one label for each argument tokens. Thus, a vector will record a set of labels as the best.

In general, through the joint unconstrained model, it will return a trigger label and a set of argument label that give the maximum score and are regard as the best labels. Moreover, because they have the maximum score of trigger and argument, the sum of them must be the highest and present the search routine (argmax) as a part of the perceptron algorithm.

4.2 Constrained joint model

In the second part, the joint model of trigger and argument are similar to the last problem but only limits by three constrained conditions which should be implemented at the same time. Thus, following the requirements, the pseudo-code of the unconstrained joint model search routine is shown below:

```
1.  algorithm Constrained joint model is
2.        input: a token $\boldsymbol{x}$
3.                the set of trigger's labels  $e_i, i = 1,2 \dots, n$
4.                the candidate argument tokens  $c_j, j = 1, \dots, m$
5.                the set of arguments' labels  $a^i_{c_j}, i = 1,2 \dots, n$
6.                triggers' feature functions  $triggerFeature$  (vector)
7.                arguments' feature functions  $argumentFeature$  (vector)
8.                the current weight parameter  $\boldsymbol{\lambda}$
9.        output: the best trigger label  $e'$
```

```
10.             the best argument label  $\boldsymbol{a}' = (a'_{c1}, \dots, a'_{cm})$
11.       For each labels of trigger  $e_i, i = 1,2 \dots, n$
12.             $\text{map}(e_i, score) \leftarrow \text{map}(e_i, score) + e_i \rightarrow dot(triggerFeature(\boldsymbol{x}, e_i), \boldsymbol{\lambda})$
13.       find  $map(e_{max}, \max score)$
14.       $e' \leftarrow e_{max}$
15.       If  $e' = \text{"}None\text{"}$  Then //Constraint 1
16.          For the candidate argument tokens  $c_j, j = 1, \dots, m$
17.          $\boldsymbol{a}' \leftarrow (\text{"}None\text{"}, \dots, \text{"}None\text{"})$
18.     //Constraint 3
19.     Else If  $e'$ $\boldsymbol{Not}(Regulation\ \boldsymbol{Or}\ Positive_{Regulation}\ \boldsymbol{Or}\ Negative_{Regulation})$  Then
20.          For the candidate argument tokens  $c_j, j = 1, \dots, m$
21.             For each labels of arguments  $a^i_{c_j}, i = 1,2 \dots, n$
22.                $\text{map}\left(a^i_{c_j}, score\right) \leftarrow \text{map}\left(a^i_{c_j}, score\right) + a^i_{c_j} \rightarrow$
                   $dot(argumentFeature\left(\boldsymbol{x}, a^i_{c_j}\right), \boldsymbol{\lambda})$
23.             find  $map(a^{max}_{c_j}, \max score)$
24.             If  $a^{max}_{c_j} = \text{"CAUSE"}$  Then
25.                $a'_{c_j} \leftarrow \text{"NONE"}$
26.             Else
27.                $a'_{c_j} \leftarrow a^{max}_{c_j}$
28.          $\boldsymbol{a}' \leftarrow (a'_{c1}, \dots, a'_{cm})$
29.          //Constraint 2
30.          If  $\boldsymbol{a}' = (a'_{c1}, \dots, a'_{cm})$  Not Contains "Theme" Then
31.             $a'_{c1} \leftarrow \text{"THEME"}$
32.          $\boldsymbol{a}' \leftarrow (a'_{c1}, \dots, a'_{cm})$
33.       Else
34.          For the candidate argument tokens  $c_j, j = 1, \dots, m$
35.             For each labels of arguments  $a^i_{c_j}, i = 1,2 \dots, n$
36.                $\text{map}\left(a^i_{c_j}, score\right) \leftarrow \text{map}\left(a^i_{c_j}, score\right) + a^i_{c_j} \rightarrow$
                   $dot(argumentFeature\left(\boldsymbol{x}, a^i_{c_j}\right), \boldsymbol{\lambda})$
37.             find  $map(a^{max}_{c_j}, \max score)$
38.             $\boldsymbol{a}' \leftarrow (a'_{c1}, \dots, a'_{cm})$
39.             //Constraint 2
40.             If  $\boldsymbol{a}' = (a'_{c1}, \dots, a'_{cm})$  !contains "Theme" Then
41.                $a'_{c1} \leftarrow \text{"THEME"}$
42.          $\boldsymbol{a}' \leftarrow (a'_{c1}, \dots, a'_{cm})$
43.       Return  $(e', \boldsymbol{a}')$
44.  end
```

From the pseudo-code, the major method is same, but it should be under the three limitations. In the first constraint, if a trigger's label is "None", it means that all the argument's labels of this trigger should be changed to "None". After that, there are two conditions: if the trigger's label is not about regulation that contains "Regulation, Positive_regulation and Negative_regulation"; or the event is regulation events. That involves the third constraint, and it indicates that the trigger cannot have a "CAUSE" argument unless it is related to regulation. Therefore, the code judges the argument and changes the "CAUSE" label to "NONE" when the trigger's label is not about regulation. Also, in the both section, the constraint 2 is embedded. In the nested judgments, it will seek whether at least one "THEME" appears. If it cannot find a "THEME" argument's label, it will enforce the first argument label to "THEME".

5 Problem 5: Implementation and evaluation for Problem 4

In this problem, it implements the algorithm of the constrained joint model of Problem 4. The argmax function is different when the algorithm finds the maximum score for the best labels. The results are compared with the unconstrained joint model and the separated model of trigger and argument on Problem 3. At the same time, the three models will apply the same best feature, the same iteration number 12 and other default settings.

The results of per-trigger and per-argument are compiled with the unconstrained joint model firstly. The unconstrained joint model combines the trigger, and their arguments score when they calculate the best labels. Both of them use the same argmax model, rather than separate functions. Table 3 shows the details values of trigger prediction for each model, and Table 4 is for argument. In general, the F1 score after using joint model is similar with the per-task result which means the joint model does not help the trigger prediction with these feature functions, and the highest improvement is the "Gene_expression" event that has 14.51% difference and the Precision is much improved. This phenomenon may state that the joint model helps to remove the mislabelled elements; consequently, the F1 is higher. At the same time, there are some events decreased, such as "Negative_regulation" that is descent 22.91%. Although the F1 and Precision have a lower score, the Recall is increased. The properties combining the trigger and argument of the joint model might be the reason to reduce the performance of "Negative_regulation" because this event has a complexed structure between trigger and argument. Also, the constraints will not impact the trigger correction ratio, and the values are same.

Model	Per-task (%)			Unconstrained Joint (%)			Constrained Joint (%)		
	P	R	F	P	R	F	P	R	F
Phosphorylation	50.00	90.00	64.29	45.00	90.00	60.00	45.00	90.00	60.00
Negative_regulation	24.14	77.78	36.84	7.52	94.44	13.93	7.52	94.44	13.93
Regulation	18.92	69.23	29.72	24.66	59.34	34.84	24.66	59.34	34.84
Protein_catabolism	60.00	80.00	68.57	63.16	80.00	70.59	63.16	80.00	70.59
Binding	18.37	76.25	29.61	22.71	71.25	34.44	22.71	71.25	34.44
Positive_regulation	16.21	75.44	26.69	24.41	63.60	35.28	24.41	63.60	35.28
Localization	51.51	54.84	53.12	37.25	61.29	46.34	37.25	61.29	46.34
Transcription	23.18	60.34	33.49	27.69	62.07	38.30	27.69	62.07	38.30
None	98.53	72.63	83.62	98.46	74.46	84.79	98.46	74.46	84.79
Gene_expression	35.93	91.47	51.60	52.32	89.77	66.11	52.32	89.77	66.11
Average	**22.49**	**77.17**	**34.83**	**22.59**	**73.54**	**34.56**	**22.59**	**73.54**	**34.56**

Table 3: The trigger evaluation results of the three model

In Table 4, these comparisons indicate that the argument evaluation improves rapidly, and the joint provides significant help, F1 becoming from 11.07% to 20.63%. "Theme" obtains the 9.73% improvement to provide the main contributions on F1. Moreover, from the higher Precision and the decreased Recall, the mislabel number is reduced, and more prediction is correct. Therefore, the above results indicate that the argument is influent by the joint model but the trigger is not. The reason might be that the label of the trigger is a factor of the argument label, such as the "Binding" could have more than one "Theme" and "Transcription" only have one. However, if the argument's label is known, it cannot infer the trigger's label. Further, the limitation also has an advanced performance on F1 26.22% larger than unconstrained model 5.89%. The limitations constrain some error when to predict the argument's label given the relevant trigger; so it produces more efficient on predicting argument rather than on trigger. For example, the "Cause" increases from 1.54% to 4.35%. Although the ratio is very low, the third condition keeps that the events expect regulation cannot have a chance to obtain "Cause" argument. Moreover, the second limitation prevents that the events other than "None" must be assigned a "Theme" argument, even though it does not have "Theme" label argument. In conclusion, the joint model and the additional constraints are useful on argument forecast.

Model	Per-task			Unconstrained Joint			Constrained Joint		
	P	R	F	P	R	F	P	R	F
None	99.91	83.35	90.88	99.60	94.02	96.74	99.52	96.41	97.94

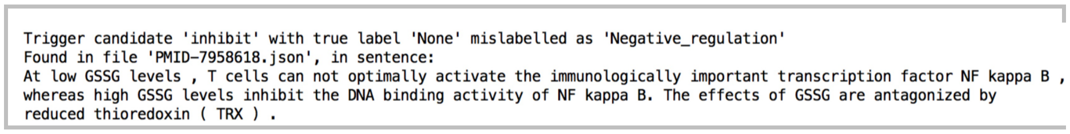
Theme	6.00	93.55	11.27	12.23	74.30	21.00	16.94	59.77	26.87
Cause	1.79	5.68	2.72	2.38	1.14	1.54	6.00	3.41	4.35
Average	**5.92**	**86.18**	**11.07**	**12.16**	**68.16**	**20.63**	**16.79**	**59.77**	**26.22**

Table 4: The argument evaluation results of the three model

6 Problem 6: Error Analysis

After events are extracted using these features and learning algorithm, we apply three post-processing rules in Constrained Joint Model to deal with limitations of our approach. One is to ensure None trigger can't have any argument. The second is to ensure trigger other than None must have at least one theme. And the third one is to ensure trigger other than None can't have Cause. Thus, this model is the best one to predict. Besides, there are other types of errors unsolved. According to errors printed, three errors were found.

1. Error due to event trigger detection(frequent)

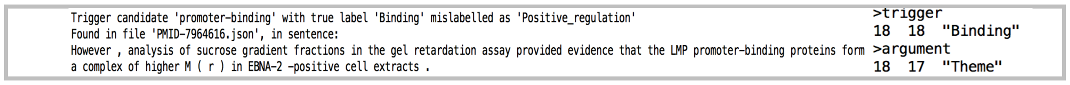


```
Trigger candidate 'inhibit' with true label 'None' mislabelled as 'Negative_regulation'
Found in file 'PMID-7958618.json', in sentence:
At low GSSG levels , T cells can not optimally activate the immunologically important transcription factor NF kappa B ,
whereas high GSSG levels inhibit the DNA binding activity of NF kappa B. The effects of GSSG are antagonized by
reduced thioredoxin ( TRX ) .
```

Figure 3. Error 1

In this case, we take "inhibit" as Negative_regulation, because there are 19 records in the training corpus, and all of them are event triggers for this event type. However, in this instance, it does not trigger an event. This error is naturally limited and will lead to recall errors, in which we do not recognize an event trigger as such, simply because we have not encountered it in the training corpus. To solve this error, more useful features can be created to take more factors into account.

2. Regulation/Positive_regulation/Negative regulation must have one theme and one cause (occasionally)

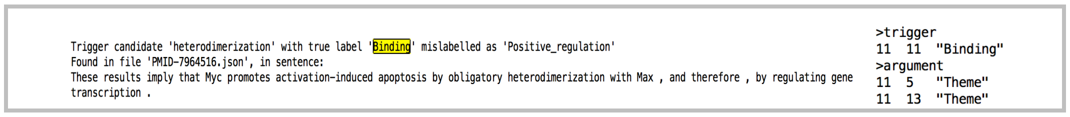


```
Trigger candidate 'promoter-binding' with true label 'Binding' mislabelled as 'Positive_regulation'                            >trigger
Found in file 'PMID-7964616.json', in sentence:                                                                                18  18  "Binding"
However , analysis of sucrose gradient fractions in the gel retardation assay provided evidence that the LMP promoter-binding proteins form >argument
a complex of higher M ( r ) in EBNA-2 -positive cell extracts .                                                                18  17  "Theme"
```

Figure 4. Error 2

In this case, if the candidate word "promoter-binding" is a Positive_regulation trigger, then it should have two arguments, including one Theme and one Cause. But in the train set, there is only one argument from "promoter-binding" to "LMP" labeled Theme.
To solve this error, the joint model can add this constraint.

3. Only Binding can have two themes (occasionally)



```
                                                                                                                       >trigger
Trigger candidate 'heterodimerization' with true label 'Binding' mislabelled as 'Positive_regulation'                 11  11  "Binding"
Found in file 'PMID-7964516.json', in sentence:                                                                        >argument
These results imply that Myc promotes activation-induced apoptosis by obligatory heterodimerization with Max , and therefore , by regulating gene   11  5   "Theme"
transcription .                                                                                                        11  13  "Theme"
```

Figure x. Error 3

In this case, the trigger word has two themes but is mislabelled as Positive_regulation. However, if there is more than one theme, it must be Binding. To solve this error, the joint model can add this constraint.

7 Problem 7: Joint Conditional Likelihood

In this problem, another training method, maximizing conditional log-likelihood is applied to data, and it uses the gradient descent to achieve the optimum solutions of vector parameters.

However, in the normal gradient descent procedure, it needs to calculate the gradient value of all the data set in very iterations. If the data set is large, it is a naive and expensive way to get the results. Therefore, the stochastic gradient descent is introduced, and both method details will be described below.

In the normal method, the conditional log-likelihood model is presented as that:

$$PO(D,\boldsymbol{\lambda}) = \sum_{(c,\boldsymbol{x}\in D)} PO(c,\boldsymbol{x},\boldsymbol{\lambda}) = \sum_{(c,\boldsymbol{x}\in D)} \log p_{\boldsymbol{\lambda}}(c|\boldsymbol{x}) = \sum_{(c,\boldsymbol{x})} s_{\boldsymbol{\lambda}}(c,\boldsymbol{x}) - s_{\boldsymbol{\lambda}}(\hat{c},\boldsymbol{x})$$

And $s_\lambda(c,x) =< f(c,x),\lambda >$, $\hat{c} = \underset{c'}{\mathrm{argmax}}\, s_\lambda(c',x)$, cis the true label and $\hat{c}$ is the predicting label; and $f(c,x)$ is the feature function results given the instance and corresponding true label; and $s_{\boldsymbol{\lambda}}(c,\boldsymbol{x})$ is the inner production. Furthermore, the maximum value of $PO(D,\lambda)$ equals to find the maximum value of $PO(c,x,\lambda)$. And then apply the gradient of this equation to find optimum results, $\nabla_\lambda PO(c,x,\lambda) = f(c,x) - f(\hat{c},x)$, and $\lambda_j = \lambda_{j-1} + \alpha \sum_{(c,x\in D)} \nabla_\lambda PO(c,x,\lambda), j = 1,..,T$. In each update step, it will compute the gradient of all instance, until achieving convergence. For example, the iteration number is 100, and it has 100 instance, which means it should calculate the gradient 10000 times. Although, the results rea the global optimum solutions absolutely, it needs running for too many times. Thus, a stochastic gradient descent is imported to reduce the calculation time. It uses random sample instances to seek the local optimum results, and this solution is the best in the global at most cases. The main difference is that when calculating the gradient, this method only obtains a random instance, which reduces the load rapidly, $\lambda_j = \lambda_{j-1} + \alpha\nabla_\lambda PO(c_i,x_i,\lambda), j = 1,..,T$. The pseudo code will be stated below:

```
1.  algorithm Joint Conditional Likelihood is
2.      input: token instances $\boldsymbol{x_i}$
3.             true label of token instances $c_i$
4.             the initial weight parameter $\boldsymbol{\lambda}$
5.      output: the suitable weight parameter $\boldsymbol{\lambda_j}$
6.
7.      Initialize $\boldsymbol{\lambda} \leftarrow [0,\dots,0]$
8.      For $j = 1,..,T$
9.        Sample instance i
10.       $\hat{c}_i \leftarrow \underset{c'}{\mathrm{argmax}}\, s_{\boldsymbol{\lambda}}(c',\boldsymbol{x})$
11.       $\nabla_{\boldsymbol{\lambda}} PO(c_i,\boldsymbol{x_i},\boldsymbol{\lambda}) \leftarrow f(c_i,\boldsymbol{x_i}) - f(\hat{c}_i,\boldsymbol{x_i})$
12.       If $\nabla_{\boldsymbol{\lambda}} PO(c_i,\boldsymbol{x_i},\boldsymbol{\lambda})! \leftarrow 0$ Then
13.         $\boldsymbol{\lambda_j} \leftarrow \boldsymbol{\lambda_{j-1}} + \alpha\nabla_{\boldsymbol{\lambda}} PO(c_i,\boldsymbol{x_i},\boldsymbol{\lambda})$
14.       Else
15.         Break
16.     Return $(\boldsymbol{\lambda_j})$
17. end
```

From the pseudo code, it describes the calculation procedures with the stochastic gradient descent method to obtain the optimum weight parameter vector. Firstly, it initializes the weight, and then uses a loop for iteration. In each iteration, it reloads an instance randomly and find the predicting label that can achieve the highest value of inner production. After that, the gradient can be gained by the binary results' difference between the true label and predict label. If the difference is not 0, the last weight values will add the production of learning ratio and the difference. On the other hand, it is convergence and that parameter is the best one.

References

[1] Kilicoglu H, Bergler S. EFFECTIVE BIO - EVENT EXTRACTION USING TRIGGER WORDS AND SYNTACTIC DEPENDENCIES[J]. Computational Intelligence, 2011, 27(4): 583-609.

[2] Collins M. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms[C]//Proceedings of the ACL-02 conference on Empirical methods in natural

language processing-Volume 10. Association for Computational Linguistics, 2002: 1-8.

[3] Björne, Jari, et al. "Extracting complex biological events with rich graph-based feature sets." Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing: Shared Task. Association for Computational Linguistics, 2009.

[4] Riedel, Sebastian, et al. "Model combination for event extraction in BioNLP 2011." Proceedings of the BioNLP Shared Task 2011 Workshop. Association for Computational Linguistics, 2011.

[5] De Marneffe, Marie-Catherine, and Christopher D. Manning. Stanford typed dependencies manual. Technical report, Stanford University, 2008.