COMPM083: Statistical Natural Language Processing - Assignment 2

Sergiu Tripon
Department of Computer Science
University College London
Gower Street, London, WC1E
sergiu.tripon.15@ucl.ac.uk

Santiago Gonzalez
Department of Computer Science
University College London
Gower Street, London, WC1E
hernan.toral.15@ucl.ac.uk

Archie Norman
Department of Computer Science
University College London
Gower Street, London, WC1E
archie.norman.15@ucl.ac.uk

Eduardo Litonjua
Department of Computer Science
University College London
Gower Street, London, WC1E
eduardo.litonjua.14@ucl.ac.uk

Problem 1: Implementation of Perceptron Algorithm - Evaluation Results

Table 1: Similarity on Average and Per-class evaluation comparison for Trigger extraction.

	My Trainer			**Precompiled Trainer**		
	Precision	**Recall**	**F1** Score	**Precision**	**Recall**	**F1 Score**
Average	0.526	0.665	0.587	0.526	0.665	0.587
Phosphorylation	0.75	1.0	0.857	0.75	1.0	0.857
Negative Regulation	0.690	0.603	0.644	0.690	0.603	0.644
Regulation	0.422	0.678	0.520	0.422	0.678	0.520
Binding	0.016	1.0	0.031	0.016	1.0	0.031
Positive Regulation	0.408	0.64	0.498	0.408	0.64	0.498
Localization	0.0	0.0	0.0	0.0	0.0	0.0
Transcription	0.0	0.0	0.0	0.0	0.0	0.0
None	0.914	0.849	0.880	0.914	0.849	0.880
Gene Expression	0.881	0.761	0.817	0.881	0.761	0.817

Problem 2: Implementation of Average Perceptron

The Perceptron algorithm returns the most recent version of the weight vector, whereas the Average Perceptron algorithm computes and returns the average of all instances of the weight vector[1].

The Average Perceptron algorithm can be implemented naively, this would indicate that the algorithm would be slow at run time. It maintains an average weight vector which is updated for every instance regardless of whether it was correctly classified. Maintaining and updating each version of the weight vector on every iteration is impractical and inefficient.

In order to improve the naive implementation of the Average Perceptron algorithm, we propose a solution whereby the average weight vector is not updated for every instance, but only for instances that were incorrectly classified. We do this as only incorrect classifications have an affect on vector weight changes. In addition, We can improve computational speed by calculating the average sum

of all updates once over every instance. Table 2 shows that we get the exact same values as the provided precompiled algorithm.

Table 2: Average and Per-class evaluation comparison for Trigger extraction.

	My Trainer			**Precompiled Trainer**		
	Precision	**Recall**	**F1** Score	**Precision**	**Recall**	**F1 Score**
Average	0.210	0.590	0.310	0.210	0.590	0.310
Phosphorylation	0.000	0.000	0.000	0.000	0.000	0.000
Negative Regulation	0.679	0.571	0.620	0.679	0.571	0.620
Regulation	0.444	0.571	0.500	0.444	0.571	0.500
Binding	0.031	1.000	0.060	0.031	1.000	0.060
Positive Regulation	0.136	0.920	0.238	0.136	0.920	0.238
Localization	0.000	0.000	0.000	0.000	0.000	0.000
Transcription	0.235	0.307	0.266	0.235	0.307	0.266
None	0.984	0.512	0.673	0.984	0.512	0.673
Gene Expression	0.787	0.357	0.492	0.780	0.350	0.490

Problem 3: Feature Engineering and Evaluation [2][3][5]

We develop two different sets of features for trigger and argument extraction tasks, and evaluate the results of running two classification models with the best features to maximize its predictions. Table 3 reports the parameters used for running the algorithms.

Table 3: Parameters for Document extraction and Algorithm Setup

Training/Sampling			**Perceptron Model**	
Train Docs	Training Ratio	Trigger/Arg Subsampling	No. of Iterations	Learning Rate
1000	80%	0.02/0.008	10	1.0

The average evaluation results of the extraction task for the *Naive Bayes (MLE) Algorithm* are presented on Table 4, while Table 5 and Table 6 shows the performance of each of the features in the training.

Table 4: Evaluation results of Naive Bayes algorithm for Trigger and Argument Extraction (average).

Trigger Extraction			**Argument Extraction**		
Precision	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
0.201	0.833	0.324	0.056	0.858	0.106

Table 5: Best set of Trigger Features for the Naive Bayes algorithm, and their impact on weight.

	Before (no features)			**After (with feature)**		
Feature	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
label bias	0.002	0.020	0.003	0.033	0.339	0.060
first trigger word	0.002	0.020	0.003	0.170	0.847	0.284
trigger dictionary starts with	0.002	0.020	0.033	0.001	0.019	0.003
first mention length	0.002	0.020	0.033	0.013	0.093	0.024

Table 6: Best set of Argument Features for the Naive Bayes Algorithm, and their impact on weight.

	Before (no features)			After (with feature)		
Feature	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
label bias	0.010	0.880	0.020	0.010	0.880	0.020
is protein_first trigger word	0.010	0.880	0.020	0.034	0.743	0.065
first argument pos	0.010	0.880	0.020	0.022	0.831	0.043
first token of event pos	0.010	0.880	0.020	0.012	0.486	0.024
first mention length	0.010	0.880	0.020	0.008	0.341	0.015

Perceptron Algorithm

Evaluation results for the *Perceptron Algorithm* are presented on Table 7, whilst Table 8 and Table 9 show the performance of each of the features in the training.

Table 7: Evaluation results of Perceptron algorithm for Trigger and Argument Extraction (average).

Trigger Extraction			**Argument Extraction**		
Precision	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
0.305	0.777	0.438	0.121	0.837	0.212

Table 8: Best set of Trigger Features for the Perceptron Algorithm, and their impact on weight.

	Before (no features)			After (with feature)		
Feature	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
label bias	0.002	0.020	0.003	0.019	0.201	0.035
first trigger word/pos/stem	0.002	0.020	0.003	0.176	0.812	0.290
first trigger length	0.002	0.020	0.003	0.063	0.383	0.109
left/right pos on nouns/verbs	0.002	0.020	0.003	0.002	0.025	0.004
none trigger event	0.002	0.020	0.003	0.002	0.020	0.004
none trigger event to	0.002	0.020	0.003	0.002	0.022	0.004
token bigram on the left/right	0.002	0.020	0.003	0.084	0.610	0.148
token starts with/contains ”-”	0.002	0.020	0.003	0.004	0.041	0.007
trigger dictionary starts/ends with	0.002	0.020	0.003	0.021	0.220	0.039
first mention length	0.002	0.020	0.003	0.024	0.164	0.042
distance to left/right mention	0.002	0.020	0.003	0.025	0.242	0.045
dep head/mod	0.002	0.020	0.003	0.029	0.269	0.053

Features choice justification

Trigger Features

1. **label bias** - A default feature for the bias term for non representative instances.
2. **first trigger word** - Feature for most common words as trigger candidates.
3. **first trigger pos/stem** - Observing the first trigger word feature provided, a feature template was created to evaluate the first candidates part of speech and stem to the trigger label in order to find out if it learns specific patterns within the data. Moreover, the feature improves recall.

Table 9: Best set of Argument Features for the Perceptron Algorithm, and their impact on weight.

	Before (no features)			After (with feature)		
Feature	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
label bias	0.010	0.880	0.020	0.010	0.880	0.020
first argument word/pos/stem	0.010	0.880	0.020	0.028	0.774	0.054
is protein_first trigger word	0.010	0.880	0.020	0.041	0.786	0.079
first token of event pos	0.010	0.880	0.020	0.017	0.641	0.033
is protein_first	0.010	0.880	0.020	0.010	0.880	0.020
unigram/bigram on the left/right using begin	0.010	0.880	0.020	0.028	0.790	0.055
unigram/bigram on the left/right using event.begin	0.010	0.880	0.020	0.021	0.648	0.040
distance between event and event.begin	0.010	0.880	0.020	0.013	0.722	0.027
none coming/going arguments due to no dependencies	0.010	0.880	0.020	0.010	0.880	0.020
dep head/mod	0.010	0.880	0.020	0.014	0.770	0.027

4. **first trigger length** - The feature evaluates the length of common words to improve the score in altogether with other word-based features. It has been found that it improves recall.
5. **none trigger event** - The feature helps to classify None triggers based on the POS part of them. A common pattern of POS on candidate and direct neighbors where found during the data analysis task, and the proposed feature improved the precision of the results.
6. **none trigger event to** - Similar to the previous one, the feature helps to minimize the misclassification of None triggers by modeling a common syntactic pattern of verbs and preposition.
7. **left/right pos on nouns/ verbs** - Precision is improved by taking care of common patterns where the POS of not None trigger candidates is whether Noun or Verb, and including the patterns of its direct neighbors as well.
8. **token bigram on the left/right** - This feature template is a one step further from the first trigger word feature template provided. It Evaluates a bigram of the current token and its left or right neighbours, helping to improves the learning process as it provides more knowledge about whats before and after the current token. Therefore, the prediction of whether the current token is a trigger label or not becomes more precise.
9. **trigger starts with/contains "-"** - As a result of performing extensive data analysis on the data provided,a number of patterns have emerged:
   (a) High probability of tokens starting with - to be Trigger events.
   (b) High probability of tokens containing - to be proteins (mentions).
10. **trigger dictionary starts/ends with** - The feature finds common patterns between trigger label and candidate word segments of words containing "-".
11. **first mention length** - As mentions (proteins) can also be triggers, a feature template was created in order to find out how many mentions are in one sentence as well as evaluate those mentions to the trigger label. A high number of mentions in a sentence, would mean a high probability that those mentions are in fact trigger labels. Adding this feature to the feature set, improved the predictions precision significantly.
12. **distance to left/right mention** - As mentions (proteins) can also be triggers, a feature template was created in order to evaluate the probability of a mention being a trigger based

on the distance to the left and right mention. This calculation has improved precision dramatically.

13. **dep head/mod** - Similarly to the feature template first trigger word (feature 2), a feature template based on dependencies was created in order to evaluate the dependency labels to the trigger labels and observe whether there are any similarities and how the model learning process behaves. Based on the result, the predictions precision increased.

Argument Features

1. **label bias** - A default feature to control the zero predicted candidates.
2. **first mention length** - Feature used for both Trigger and Argument Extraction. See justification in the Trigger Features section above.
3. **first argument word/pos/stem** - Having access to the first argument, enabled the ability to create a feature which evaluates the first argument word, pos and stem in order to find out if there is a match and the first argument word is a valid argument.
4. **is protein_** - By identifying whether the argument head word is a protein, the chances of that word to be involved in an argument extraction increase, as proteins have a high probability of both being triggers and begin involved in an argument.
5. **first token of event pos** - This feature is similar to the trigger feature first trigger pos/stem and the justification for it would be the same. This feature slightly improves precision.
6. **is protein_first trigger word** - The provided feature evaluates a bigram to the argument label and improves precision slightly.
7. **unigram/bigram on the left/right using begin/event.begin** - Similarly to the token bigram on the left/right trigger feature, this feature evaluates two aspects:
   (a) **unigrams** - the token to the left or the right of the first token of event
   (b) **bigrams** - the token to the left or the right of the first token of event and the first token of event

   Using the unigrams and bigrams feature, the model learn the left and right neighbours of the head token of the event, which improves its prediction of the source and destination triggers which are related to one another through an argument. Using these features has a significant impact on the evaluation results, especially on precision.
8. **distance between event and event.begin** - By applying this feature, the distance between the start of the candidate and the start of the parent event is calculated for evaluation.
9. **none coming/going arguments due to no dependencies** - The feature evaluates the total number of associated syntactic dependencies in head and modifier words of the candidate. It allow us to model None events where no dependencies are found in context.
10. **dep head/mod** - Feature used for both Trigger and Argument Extraction. See justification in the Trigger Featuers section above.

Problem 4: Joint Perceptron[4]

For a given token **x** and candidate argument tokens $c_1 ... c_m$, let $\mathbf{e} \in \tau \rightarrow (\tau \cup None)$ be the label of the trigger, and $\mathbf{a} = a_{c_1}, ..., a_{c_m} : a_{c_j} \in \mathcal{R} \rightarrow (\mathcal{R} \cup None)$ the labels of the argument candidates. Then the fully factorized model scores the joint model of **e** and **a** by using formula 1:

$$\mathbf{s}(\mathbf{e}, \mathbf{a}; \mathbf{x}, \boldsymbol{\lambda}) = \mathbf{s}_\tau(\mathbf{e}; \mathbf{x}, \boldsymbol{\lambda}) + \sum_j \mathbf{s}_R(\mathbf{a}_{c_j}; \mathbf{x}, \boldsymbol{\lambda}) \tag{1}$$

Unconstrained Joint Model

Algorithm 1 shows the pseudocode for computing the an unconstrained joint model. Starting with a candidate token **x**, learning weight vector λ and feature vectors ϕ_t, ϕ_c, the *argmax* method gets the maximum score for trigger label and the associated arguments labels in isolation, and as a dual

decomposed manner. Finally, it returns the best joint scores for trigger and arguments on candidate token **x**.

Algorithm 1: Joint Unconstrained Model

Input: $\mathbf{x}$: Candidate $\boldsymbol{\lambda}$: weights
ϕ_t: Trigger Features ϕ_c: Argument Features
Result: $\mathbf{s}(\mathbf{e}, \mathbf{a}; \mathbf{x}, \boldsymbol{\lambda})$
$\boldsymbol{\rho}_\lambda(\mathbf{x}_t|\tau) \leftarrow \langle \boldsymbol{\phi}_t(\tau, \mathbf{x}_t), \boldsymbol{\lambda} \rangle$
$\widehat{\mathbf{e}} \leftarrow argmax_e \boldsymbol{\rho}_\lambda(\mathbf{x}_t|\tau)$
$\mathbf{a}_c \leftarrow argument\_candidates(\mathbf{x})$
foreach $\boldsymbol{a}_{c_j}$ *in* $\boldsymbol{a}_c$ **do**
  $\boldsymbol{\rho}_\lambda(\mathbf{a}_{c_j}|\mathcal{R}) \leftarrow \langle \boldsymbol{\phi}_c(\mathcal{R}, \mathbf{a}_{c_j}), \boldsymbol{\lambda} \rangle$
  $\widehat{\mathbf{a}_{c_j}} \leftarrow argmax_a \boldsymbol{\rho}_\lambda(\mathbf{a}_{c_j}, \mathcal{R})$
return $\mathbf{s}_\tau(\mathbf{e}; \mathbf{x}, \boldsymbol{\lambda}) + \sum_j \mathbf{s}_R(\mathbf{a}_{c_j}; \mathbf{x}, \boldsymbol{\lambda})$ `/* returns best` $\widehat{\mathbf{e}}$ `and` $\widehat{\mathbf{a}_{c_j}}$ `vector */`

Constrained Joint Model

For the Constrained Joint Model, we add up three constraits that are evaluated to get the *argmax* value for arguments. Inside Algorithm 2, the method *argmaxFcn* is responsible to compute the predictions for the candidate arguments.

Algorithm 2: Joint Constrained Model

Input: $\mathbf{x}$: Candidate $\boldsymbol{\lambda}$: weights
ϕ_t: Trigger Features ϕ_c: Argument Features
Result: $\mathbf{s}(\mathbf{e}, \mathbf{a}_c; \mathbf{x}, \boldsymbol{\lambda})$
initialization:
$argmax_{score} \leftarrow -\infty, argmax_{position} \leftarrow 0$
Method `argmaxFcn(`$\mathcal{R}, \widehat{\boldsymbol{e}}, \boldsymbol{a}_c, \boldsymbol{\lambda}, \boldsymbol{\phi}_c$`)`
  $\widehat{a_c} \leftarrow []$
  foreach $\boldsymbol{a}_{c_j}$ *in* $\boldsymbol{a}_c$ **do**
    $\boldsymbol{\rho}_\lambda(\mathbf{a}_{c_j}|\mathcal{R}) \leftarrow \langle \boldsymbol{\phi}_c(\mathcal{R}, \mathbf{a}_{c_j}), \boldsymbol{\lambda} \rangle$
    $\widehat{\mathbf{a}_{c_j}} \leftarrow argmax_a \boldsymbol{\rho}_\lambda(\mathbf{a}_{c_j}, \mathcal{R})$
    if $\widehat{\boldsymbol{a}_{c_j}^{score}} > argmax_{score}$ **then**
      $argmax_{score} \leftarrow \mathbf{a}_{c_j}^{score}$
      $argmax_{position} \leftarrow j$
  if *isNoneEvent(*$\widehat{\boldsymbol{e}}$*)* **then** `/* None events cannot have arguments */`
    foreach $best_{a_{c_j}}$ *in* $\boldsymbol{best}_{a_c}$ **do**
      $\mathbf{best}_{a_{c_j}} = None$
  else if *Not(isRegulation(*$\widehat{\boldsymbol{e}}$*))* **then** `/* Only regulation event have Cause args */`
    foreach $best_{a_{c_j}}$ *in* $\boldsymbol{best}_{a_c}$ **do**
      $best_{a_{c_j}} =$ **if** *isCause(*$best_{a_{c_j}}$*)* **then** None **else** $best_{a_{c_j}}$
  else if *Not(hasTheme(*$\boldsymbol{best}_{a_c}$*))* **then** `/* events != None can have 1+ Theme */`
    $best_{a_{c_{argmax_{position}}}} = Theme$
  return $\widehat{\mathbf{a}_c}$
$\boldsymbol{\rho}_\lambda(\mathbf{x}_t|\tau) \leftarrow \langle \boldsymbol{\phi}_t(\tau, \mathbf{x}_t), \boldsymbol{\lambda} \rangle$
$\widehat{\mathbf{e}} \leftarrow argmax_e \boldsymbol{\rho}_\lambda(\mathbf{x}_t|\tau)$
$\mathbf{a}_c \leftarrow argument\_candidates(\mathbf{x})$
$\mathbf{best}_{a_c} \leftarrow$ `argmaxFcn(`$\mathcal{R}, \widehat{\boldsymbol{e}}, \boldsymbol{a}_c, \boldsymbol{\lambda}, \boldsymbol{\phi}_c$`)`
return $\mathbf{s}_\tau(\mathbf{e}; \mathbf{x}, \boldsymbol{\lambda}) + \sum_j \mathbf{s}_R(\mathbf{a}_{c_j}; \mathbf{x}, \boldsymbol{\lambda})$ `/* returns best` $\widehat{\mathbf{e}}$ `and` $\mathbf{best}_{a_c}$ `vector */`

Problem 5: Implementation and Evaluation for Problem 4

The best set of trigger and argument features have been applied to the four algorithms discussed: Naive Bayes, Perceptron, Joint Unconstrained and Joint Constrained.

After evaluating the results of running the two joint algorithms and the previous Naive Bayes and Perceptron models using the best set of developed (trigger and argument) features, we can say that there is not one algorithm that scored the highest in both trigger and argument extraction tasks. Table 10 presents the evaluation results for all the algorithms using the same configuration as proposed in Problem 3. The Perceptron Algorithm obtained a better prediction for Trigger extraction, while the Joint Unconstrained model predicted better in the Argument extraction task.

Table 10: Evaluation results comparison of the Naive Bayes, Perceptron, Joint Unconstrained and Joint Constrained algorithms.

	Trigger Extraction			**Argument Extraction**		
Algorithm	**Precision**	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
Naive Bayes	0.201	0.833	0.324	0.056	0.858	0.106
Perceptron	0.305	0.777	0.438	0.121	0.837	0.212
Joint Unconstrained	0.220	0.820	0.347	0.141	0.758	0.237
Joint Constrained	0.220	0.820	0.347	0.143	0.587	0.230

The Joint Unconstrained model scored the highest evaluation values for the Argument extraction task in comparison to the Joint Constrained model which we expected to score better. the Perceptron algorithm scored higher than both Joint algorithms in the Trigger Extraction task due to the developed features performs better in the per-task model boosting Precision metrics, however the Joint algorithms had better Recall metrics. On the contrary, Naive Bayes achieved the highest Recall score on the same task. Based on these conclusions we know that Precision plays a large part in determining the best prediction algorithm on a extraction task. On the other hand, The Joint Constrained Model scored the best Precision metric in the Argument extraction task but also the worst Recall Metric which would suggest why the model's F1 score did not come out as the best.

All in all, we have seen both expected and unexpected outcomes where The Naive Bayes scoring model reached the lowest metrics and the Joint Constrained algorithm scored higher than both the previous and Perceptron algorithms as supposed. However, we had not anticipated the Joint Unconstrained model as better predictioner than the Constrained Model in the Argument extraction task.

Problem 6: Error Analysis

The best model we have developed so far is the Perceptron Algorithm. Table 11 and Table 12 indicate frequencies for triggers and arguments where our model misclassified within the dev set.

Table 11: Most frequent errors given by the Perceptron algorithm for Trigger Extraction. (B = Binding, GE = Gene Expression, L = Localisation, -R = Negative Regulation, N = None, P = Phosphorylation, +P = Positive Regulation, PC = Protein Catabolism, R = Regulation, T = Transcription)

	Trigger mislabelled as									
Gold	**B**	**GE**	**L**	**-R**	**N**	**P**	**+R**	**PC**	**R**	**T**
B	N/A	233	N/A	257	1207	N/A	63	N/A	72	137
-R	64	76	N/A	N/A	1380	N/A	815	64	163	157
N	18385	12811	1612	26920	N/A	1270	70043	1032	26754	17711
+R	157	67	N/A	538	6319	N/A	N/A	N/A	778	531
R	164	N/A	N/A	N/A	869	N/A	2285	95	N/A	60

Table 12: Most frequent errors given by the Perceptron algorithm for Argument Extraction.

	Argument mislabelled as		
Gold	**Cause**	**Theme**	**None**
None	32728	161881	N/A
Theme	785	N/A	2896
Cause	N/A	884	826

These triggers were mislabelled as Regulation and Positive Regulation when they should have been labelled as None (triggers in bold font):

*The glucocorticoid receptor was able to further enhance IL-4 -induced gene transcription through the **action** of Stat6.*

*This study tested the hypothesis that anti-LPS Abs neutralize endotoxin by **blocking** cellular uptake through mCD14.*

These arguments were mislabelled as Theme and Cause when they should have been labelled None:

*The complex on the proximal site (**NF kappa B2**) appears to be composed of p50 and relB.*

*Coactivation by **OCA-B** : definition of critical regions and synergism with general cofactors.*

By using the Perceptron algorithm for Argument extraction, we find instances where the argument is misclassified because it evaluates both the triggers and the arguments separately whereas a joint model considers the connection between them for prediction and therefore it can reduce this type of errors by analyzing the full context on every iteration.

Adding more constraints to the joint constrained model will help remedy specific argument errors i.e. a constraint for binding triggers to accept only Theme events to mentions. In addition, some improvements can be done by expanding the work carried in our data analysis work flow (see attached) which would help us to identify better and more specific high impact features for regulation triggers and corresponding theme and cause events which seems some challenging to model.

Problem 7: Joint Conditional Likelihood

We are looking to maximize the conditional log-likelihood of the data by using the stochastic gradient descent/ascent method.

Assuming that $p_{\boldsymbol{\lambda}}(c_i|\mathbf{x}_i)$, takes the following form:

$$exp\Big(\sum_{k_i} \lambda_{k_i}\phi_{k_i}(\mathbf{x}_i, c_i)\Big) \tag{2}$$

The index k_i represents the different values that $\boldsymbol{\lambda}$ can take with respect to the pair $(\mathbf{x}_i, c_i)$. The objective then takes the following form:

$$\sum_{(\mathbf{x}_i,c_i)} \sum_{k_i} \lambda_{k_i}\phi_{k_i}(\mathbf{x}_i, c_i) \tag{3}$$

Partially differentiating with respect to each component of λ_{k_i} will yield a sum of unique feature functions with non-unique arguments $(\mathbf{x}_i, c_i)$. When c_i is chosen for a given $\mathbf{x_i}$ such that all the associated feature functions ϕ_{k_i} are 0, then we have reached the optimum for the objective function.

In terms of computation, the naive and prohibitively expensive way of doing this is simply by iterating through every possible pair of $(\mathbf{x}_i, c_i)$ - i.e. the entire training set and deriving all feature functions for those pairs. A more tractable method could be reducing the total number of c_i given $\mathbf{x_i}$ (i.e. by noting that associated candidate argument tokens must exist in the same sentence as the source token).

References

[1] Michael Collins. "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms". In: *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics. 2002, pp. 1–8.

[2] David McClosky, Mihai Surdeanu, and Christopher D Manning. "Event extraction as dependency parsing". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics. 2011, pp. 1626–1635.

[3] Makoto Miwa et al. "Event extraction with complex event classification using rich features". In: *Journal of bioinformatics and computational biology* 8.01 (2010), pp. 131–146.

[4] Sebastian Riedel and Andrew McCallum. "Robust biomedical event extraction with dual decomposition and minimal domain adaptation". In: *Proceedings of the BioNLP Shared Task 2011 Workshop*. Association for Computational Linguistics. 2011, pp. 46–50.

[5] Sebastian Riedel et al. "Model combination for event extraction in BioNLP 2011". In: *Proceedings of the BioNLP Shared Task 2011 Workshop*. Association for Computational Linguistics. 2011, pp. 51–55.