Feature Extraction for Token Based Word Alignment for Question Answering Systems

Lokesh Kumar Sharma[1], Namita Mittal[2], Anubha Aggarwal[3]

[1] Dept of CSE, Galgotias College of Engg and Tech, Greater Noida, UP, India

[2] Dept of CSE, Malaviya National Institute of Technology Jaipur, RJ, India

[3] Dept of CSE, Shri Mata Vaishno Devi University, Katra, India

lokesh.gbu@gmail.com, nmittal.cse@mnit.ac.in, anubhaaggarwal95@gmail.com

Abstract. Mapping between the source words and the target words in a set of parallel sentences are a crucial part of Question Answering (QA) systems. If an accurate aligner is used in QA systems then the efficiency of these systems also gets increased. We purpose the aligner which despite using very less lexical resources gives very good results in terms of precision, recall and F1. Previous aligners either uses more lexical resources or uses very less lexical resources. Hence, we have used POS TAG and WordNet as lexical resources. But some words whose meaning we may not know but these occur in a similar distribution and by observing their distribution these words are similar. Consider two sentences "Lambodar is the son of Parvati" and "Ganesha is the son of Parvati". Here we will not find the meaning of Lambodar and Ganesha in Wordnet but since they have similar distributions so they should be aligned. For these words, we used Distribution Similarity Feature in our word aligner. This distributional similarity helps our aligner in broader coverage of words. Previous aligners were having recall in the range of 75-86 but this aligner has recall in the range of 88.4-93.3. Similarly, Exact match of previous aligners was in the range of 21-35.3 but the proposed aligner's exact match range is 46.1-58.6. Similarly F-measure and precision have also increased.

Keywords. Structural feature, question alignment, feature score, alignment score.

1 Introduction

Question answering is the task of returning a particular piece of information to the user in response to a question. In this task, it compares words in given question with words in a paragraph. Basically, it tries to align question with lines in a paragraph and based on alignment, the score is given to each line and the line with the highest score is returned as an answer to the question. In short, QA systems efficiency relies on the efficiency of Word Aligner used. In short an aligner with good precision, recall and f-measure, the exact-match rate is required. Chambers [2] in 2007 purposed Stanford RTE Aligner. It was a token based aligner and used a rich set of features for alignment scoring. But it's recall was 75.8%, F-measure was 79.1%, Precision was 82.7% on RTE2 data-set. MacCartney [8] in 2008 purposed Manli Aligner. It was the first phrase-based aligner. It also used a feature-based scoring system. Its recall was 85.3%, F-measure was 85.3%, the exact-match rate was 21.3% and precision was 85.4%. It used around 5 GB of lexical resources i.e a huge amount of lexical resources. Manli aligner was improved by Thadani and Mc Keown [10] in 2011. They used ILP for decoding due to which

ISSN 1405-5546
doi: 10.13053/CyS-22-4-3070

speed and efficiency of Manli aligner got increased. But it's recall was 86.2%, F-measure was 87.8%, the exact-match rate was 33.0%, precision was 89.5%. Xuchen Yao [12] in 2013 purposed Jacana Token-based Aligner. It used Conditional Random Field to decide the best alignment with features drawn from source and target sentence. It used very less lexical resources and its results in terms of efficiency and speed were also good. But its best recall was 86.6%, the maximum exact-match rate was 35.3%, best f-measure was 88.6% and best precision was 95.4%.

In this paper, we describe a token-based word-aligner. It uses various features for scoring like distributional similarity feature, String-similarity feature. Results show that when we run our aligner on the RTE2 Dataset, our aligner gives maximum recall of 93.3, maximum exact-rate of 58.6%, maximum F-measure of 93.8%, maximum precision of 96.6%. This states that our aligner is giving a state-of-art performance in terms of efficiency and if used for a question answering system it can increase the efficiency of Question Answering System. But the drawback is that it's aligning one sentence pair per second and needs improvement in speed.

2 Related Work

Manli Aligner [8] (Bill MacCartney et al., 2008) was the first aligner to purpose how to align hypothesis and premise of English sentence pair. It was proposed by MacCartney [7] in 2008. It used perceptron learning for this. It was the first aligner which supported phrase based alignment of arbitrary lengths. It used around 5 GB of lexical resources and took around 2 sec per alignment. Thadani and McKeown [10] in 2011 optimized Manli aligner by decoding via Integer Linear Programming(ILP). ILP lead to an increase of Manli alignment speed. They added extra syntactic constraints due to which alignment precision, recall, F1 got improved. After this came Jacana token based Aligner. It is the first open-source aligner. Jacana aligner is discriminatively trained monolingual word aligner. It models a many-to-one alignment from source to target. It uses first-order Conditional Random

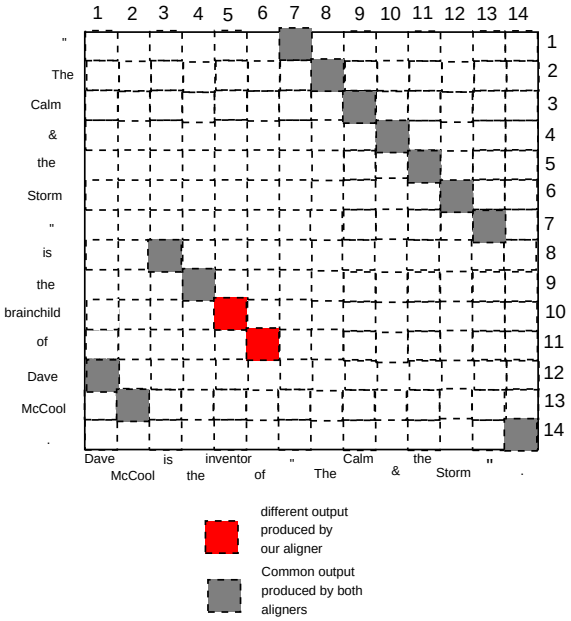

Fig. 1. Figure shows comparision of output by Jacana and proposed aligner

Field to globally decode the best alignment. Best possible sequence decision is taken on the basis of certain features like String Similarity Feature, POS Tag feature, Positional Feature, WordNet Feature, Contextual Feature, Distortion Feature. It uses just part-of-speech tags and Wordnet as lexical resources and gives good precision and fast results than improved Manli Aligner (proposed by Thadani and McKeown).

These aligners are based on supervised methods. Some aligners used unsupervised methods also. These include the work of Wan and Mark [11] (2010) who extended the work of McCallum et al. [9] (2005) and modeled alignment as latent variables. Heilman and Smith in 2010 used tree kernels to search for the alignment that yields the lowest tree edit distance. Other tree or graph matching work for alignment includes that of (Blunsom et al., 2006 [1]; John D et al., 2001 [6]; Kevin et al., 2010 [4]; Marneffe et al., 2006 [3]; Stephen Wan et al., 2006 [11]).

Fig 1 shows comparison of output produced by our aligner and Jacana Aligner. It shows that because of distributional similarity feature our

aligner aligns word brainchild and inventor and of and of which were not aligned by Jacana Aligner. Here blocks in gray signify the alignment that is produced by both aligners and words in red signify the alignment output that is given by our aligner but not Jacana Aligner. The proposed aligner is matching the text on both lexical and syntactic together but semantic is done separately.

3 Proposed Alignment

In this section, we describe a model for our aligner. Our model uses feature-based alignment. For feature-based alignment various basic and proposed features [5] are required to extract. These features include lexical, syntactic, semantic and structural features. Further, these features are used to calculate a feature form score which is useful to measure feature based similarity. Our aligner is highly influenced by Jacana token based Aligner. Experiments show that for word aligner, using distributional features and string similarity features helps in increasing efficiency of aligner.

3.1 Model

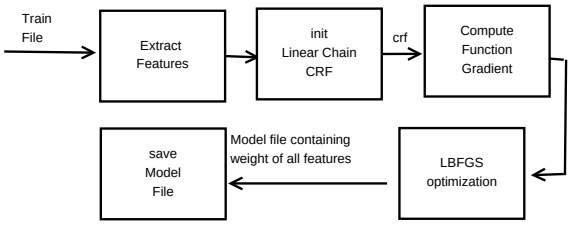

Fig. 2. Training model

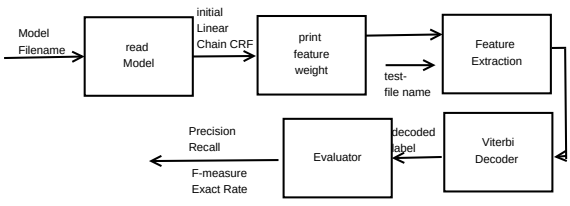

Fig. 3. Testing model

Given a source sentence s of length M, and a target sentence t of length N, the alignment from s to t is a sequence of target word indices a, where $a_i \in$ [1,M] $\in$ [0, N]. We specify that when $a_i \in = 0$, source word s_i is aligned to a NULL state, i.e., deleted. This models a many-to-one alignment from source to target: multiple source words can be aligned to the same target word, but not vice-versa. One-to-many alignment can be obtained by running the aligner in the other direction. The probability of alignment sequence a conditioned on both s and t is shown in equation (1):

$$p(a|s,t) = \frac{exp(\sum_{i,k} \lambda_k f_k(a_{i-1}, a_i, s, t))}{Z(s,t)}. \quad (1)$$

This assumes a first-order Conditional Random Field [9]. Since the word alignment task is evaluated over F1, instead of directly optimizing it, we choose a much easier objective [4] and add a cost function to the normalizing function $Z(s,t)$ in the denominator is shown in equation (2):

$$Z(s,t) = \sum_{\hat{a}} (exp(\sum_{i,k} \lambda_k f_k(a_{i-1}, a_i, s, t)) + cost(a_y, \hat{a})), \quad (2)$$

where a_y is the true alignment. The value of this special function $cost(a_y, a)$ can be viewed as the features that encourage decoding to be consistent with true labels. It is only computed during training in the denominator because in the numerator $cost(a_y, a_y) = 0$. The hamming cost is used in practice without learning the weights (i.e., uniform weights). The more in-consistence there is between a_y and a, the more penalized is the decoding sequence a through the cost function.

Figure 2 and 3 shows training and testing model. Training phase focuses on producing a model file. The model file contains a weight for each feature. This model file will be used in the testing phase. In training phase firstly features are extracted from training data-set. Then a Linear Chain CRF is initialized. After this Compute Function Gradient function on CRF object is called. This function helps in assigning weight to each feature. After this LBFGS optimization algorithm is called. This optimizes feature weight.

Finally, the model is saved for the testing purpose.In the testing phase model file that we obtained during the training phase is read and

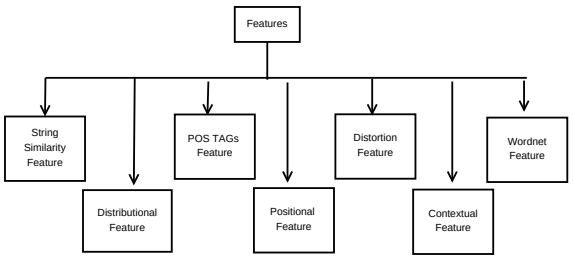

Fig. 4. Features used in aligner

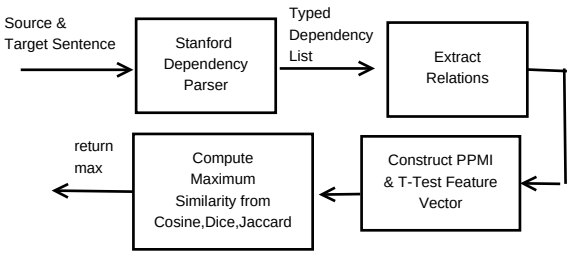

Fig. 5. Distributional Similarity Overview

from that Linear Chain, CRF is initialized. Feature weights are also printed on the output screen. After this features are extracted from testing data. Viterbi Decoder is used to decode the best alignment sequence. Decoded label are given to the evaluator and then evaluator evaluates based on the comparison of desired output sequence and obtained output sequence. Finally, precision, recall, f-measure, exact match rate is printed on the screen.

3.2 Feature Design

In fig. 4 the features that we have used in our aligner are shown together.

Similarity Feature: These include features like IdenticalMatch, IdenticalMatchIgnoreCase, MaxMatch. Identical Match is when two strings are completely same. Identical Match Ignore Case is when these condition matches: i) stem words of two strings match or ii) if a word A's stem word completely contains another word B's stem word and index within the stem of word A of the first occurrence of the stem of word B should be 0. For example words like random and randomly or makes and maker or iii) If a word A is composed of two or more words combined by a hyphen (-) like supporting-actress and another word's stem match with any of stem of words obtained by splitting word A on basis of hyphen. Eg words like supporting-actress and actress or iv) If both words are composed of two or more words combined by a hyphen(-) and they are like supporting-actress-father and supporting actresses.

MaxMatch is the maximum similarity value between two words if we compare them using methods like JaroWinkler, DiceSorensen, Jaccard, normalized Levenshtein, NGRAM3, NGRAM4, normalized common_prefix, normalized common_suffix. Two words are similar by MaxMatch only if max_match value is greater than a threshold. All these methods give a value between 0 and 1 so MaxMatch value is also between 0 and 1.

String Similarity Algorithm:

Algorithm 1: String Similarity Feature

Input:
srcToken: Token of source string to be compared
tgtToken: Token of target string to be compared with
Algorithm:

```
begin
|    srcStem=stem(srcToken)
|    tgtStem=stem(tgtToken)
|    if(srcStem==tgtStem)
|    begin
|    |    addFeature(IdenticalMatchIgnoreCase,1.0)
|    |    if(srcToken==tgtToken)
|    |    begin
|    |    |    addfeature(IdenticalMatch,1.0)
|    |    end
|    else if(srcStem.indexOf(tgtStem)==0 or
|    |    tgtStem.indexOf(srcStem)== 0)
|    |    if(srcStem.length > 2 and
tgtStem.length>2)
|    |    begin
|    |    |    addFea-
ture(IdenticalMatchIgnoreCase,1.0)
|    |    end
|    else if(srcToken contains (”-”) or
|    |    tgtToken contains(”-”))
|    |    if(srcToken contains(”-”) and
```

```
|   |   !tgtToken contains("-"))
|   |   begin
|   |   |   var splitArr=srcToken.split("-")
|   |   |   if(stem of any word in splitArr
match
|   |   |   with tgtStem)
|   |   |   begin
|   |   |   |   addFeature(IdenticalMatch-
|   |   |   |   IgnoreCase,1.0)
|   |   |   end
|   |   else if (tgtToken contains("-") and
|   |   !srcToken contains("-"))
|   |   |   var splitArr=tgtToken.split("-")
|   |   |   if(stem of a ny word in splitArr
match
|   |   |   with srcStem)
|   |   |   begin
|   |   |   |   addFeature(IdenticalMatch-
|   |   |   |   IgnoreCase,1.0)
|   |   |   end
|   |   else if(tgtToken contains("-") and
|   |   srcToken contains("-") )
|   |   |   var src_splitArr=srcToken.split("-")
|   |   |   var tgt_splitArr=tgtToken.split("-")
|   |   |   if corresponding words in
src_splitArr
|   |   |   and tgt_splitArr match untill smaller(
|   |   |   src_splitArr.length-1 ,
|   |   |   tgt_splitArr.length-1)
|   |   |   begin
|   |   |   |   addFeature(IdenticalMatch-
|   |   |   |   IgnoreCase,1.0)
|   |   |   end
|   |   end
|   else if(max(JaroWinkler(srcToken,tgtToken),
|   DiceSorensen(srcToken,tgtToken), Jaccard
|   -Metric(srcToken,tgtToken),normalized_
|   Levenshtein(srcToken,tgtToken),
|   NGRAM3(srcToken,tgtToken),
|   NGRAM4(srcToken ,tgtToken),normalized
|   _common_prefix(srcToken,tgtToken),normalized
|   _common_suffix(srcToken,tgtToken))>
threshold)
|   |   addFeature(MaxMatch,max_value)
|   end
|   if((special match ,value) <-
|   special match(srcToken,tgtToken))
|   |   addFeature(special match,value)
end
```

Distributional Feature: This feature takes care of words that are similar in their distribution. For example: Kalpana is the mother of Ram, Kalpana is the mother of Shyam. Here Ram and Shyam are distributionally related. It uses PPMI and T-test as the measure of association and dice, Jaccard, cosine as the measure of similarity. Figure 5 shows an overview of this feature.

Distributional Similarity Algorithm:

Algorithm 2: Distributional Similarity

Input:
srcString: Source string to be compared
tgtString: Target string to be compared with
Algorithm:

```
begin
|   TypedDependenciesList=Parse both strings
in
|   Stanford Dependency Parser
|   for each typedDependency in Typed-
|   DependencyList
|   begin
|   |   Extract GrammaticalReal-
tion,word1,word2
|   |   from typedDependency
|   |   Make a string feature = Grammatical-
|   |   Relation+" of "+word1 // this string will
|   |   serve as feature in feature vector so
store
|   |   all these features.
|   end
|   splitsrcArr=srcString.split(string_separators)
|   splittgtArr=srcString.split(string_separators)
|   // initialFeatureVectors will contain feature-
|   // vectors of src word and tgt word.
|   for each word in splitsrcArr and splittgtArr
|   |   Make a feature vector where features
|   |   extracted above will serve as
components
|   |   of vector and their values will be count
|   |   of how many times word occurs in
|   |   relationship with feature i.e word occurs
as
|   |   word2 in feature+2.0.
|   |   if word occurs as word2 in any feture
|   |   |   add word's feture vector to
```

```
|   |   |   initialFeatureVectors
|   end
|   //From initialFeatureVectors contruct
|   //PPMIFeatureVectors and
t-TestFeatureVectors
|   //by applying PPMI and t-test formula to each
|   //entry of initialFeatureVectors.
|   for each srcword in splitsrcArr
|   begin
|   |   src_ppmi_feature_vector=
|   |   PPMIFeatureVectors.get(srcword)
|   |   src_t-test_feature_vector=
|   |   t-TestFeatureVectors.get(srcword)
|   |   for each tgtword in splittgtArr
|   |   begin
|   |   |   tgt_ppmi_feature_vector=
|   |   |   PPMIFeatureVectors.get(tgtword)
|   |   |   tgt_t-test_feature_vector=
|   |   |   t-TestFeatureVectors.get(tgtword)
|   |   |   maxValue=max(Cosine(src_ppmi_
|   |   |   feature_vector,tgt_ppmi_feature
|   |   |   _vector),Dice src_ppmi_feature_
|   |   |   vector,tgt_ppmi_feature_vector),
|   |   |   Jaccard(src_ppmi_feature_vector,
|   |   |   tgt_ppmi_feature_vector),
|   |   |   Cosine(src_t-test_feature_vector,
|   |   |   tgt_t-test_feature_vector),
|   |   |   Dice(src_t-test_feature_vector,
|   |   |   tgt_t-test_feature_vector),
|   |   |   Jaccard(src_t-test_feature_vector,
|   |   |   tgt_t-test_feature_vector))
|   |   |   addFeature("DistriSim",maxValue)
|   |   end
|   end
end
```

POS Tags Feature: This feature checks whether POS Tag of the words being compared match. If their POS match then POS_MATCH feature is added to source word else POS_NO_MATCH feature is added to source word.

Positional Feature: In this POSITION and POSITION_RELATIVE feature is added to source word. POSITION feature has feature value as abs(i-j) and POSITION_RELATIVE feature has feature value as abs(i/M -j/N) where M, N are the lengths of source sentence and target sentence respectively , i the index of source word in source sentence and j is the index of target word in target sentence.

Distortion Feature: Distortion Features measure how far apart the aligned target words of two consecutive source words are: abs$(a_m + 1 - a_{m-1})$. Special features for corner cases where the current word starts or ends the source sentence, or both the previous and current words are deleted (a transition from NULL to NULL) are also taken care of.

Contextual Feature: If a source word is getting aligned to multiple words then the best target word for source word is chosen through this feature. This feature indicates whether left or right neighbors of source word and aligned target word match or are identical, or whether pos of right or left neighbors match.

WordNet Feature: This feature checks whether the words being compared satisfy any of these relations hypernym, hyponym, synonym, derived form, entailing, causing, members of, have member, substances of, have substances, parts of, have part; or whether their lemmas match. These relations act as feature and whenever any relation is satisfied, that relation is added as feature to source word.

4 Dataset, Experimental Setup, and Results

4.1 Dataset Used

To evaluate the performance of the proposed aligner, one of the most popular publicly available Dataset is used. This standard Dataset, known as (Brockett, 2007) Dataset consists of 800 manually aligned premise and hypothesis pairs from the RTE2 data set. The length of Premises(source_sentence) on an average is 29 words and of hypotheses(target_sentence) is 11 words only. We take the premise as the source and hypothesis as the target. S2T indicates that model aligns from source to target and T2S indicates that model aligns from target to source.

4.2 Experimental Setup

4.3 Baselines

4.3.1 Stanford RTE Aligner Baseline

The Stanford aligner(Chambers et al., 2007) uses a simpler, token based alignment representation, along with a richer set of features for alignment scoring. It represents alignments as an injective map from H tokens to P tokens. Phrase alignments are not directly representable, although the effect can be approximated by a pre-processing step which collapses multi-token named entities and certain collocations into single tokens. The features used for alignment scoring include not only measures of lexical similarity, but also syntactic features intended to promote the alignment of similar predicate-argument structures.

4.3.2 MANLI Baseline

MANLI was first developed by MacCartney et al. (2008) and then improved by Thadani and McKeown (2011) with faster and exact decoding via ILP. There are four versions to be compared here: MANLI the original version. MANLI-approx. comparison version by Thadani and McKeown (2011). MANLI-exact decoding via ILP solvers. MANLI-constraint MANLI-exact with hard syntactic constraints, mainly on common "light" words (determiners, prepositions, etc.) attachment to boost exact match rate. MANLI uses an alignment representation which is intrinsically phrase-based. (Following the usage common in MT, we use "phrase" to mean any contiguous span of tokens, not necessarily corresponding to a syntactic phrase.)

4.3.3 Jacana Token Based Baseline

Jacana was developed by Yao et al. (2013a). It was a token based aligner. It gave a state-of-art performance (i.e produced results faster and improved precision, recall, F1 also) on RTE2 set. In Jacana Paper the aligner was run in two directions s2t and t2s and then results were merged with intersection and union.

We have compared our aligner with above-mentioned aligner's on the basis of P, R, F, E. P stands for Precision, R stands for Recall, F stands for F-measure, E stands for Exact rate match. We have computed results i.e trained and tested our aligner only, rest all of the values are taken from their respective papers.

4.4 Results and Discussions

Table 1. Results on 800 pairs of test data

System	P%	R%	F%	E%
Stanford RTE	82.7	75.8	79.1	-
MANLI	85.4	85.3	85.3	21.3
MANLI-approx	87.2	86.3	86.7	24.5
MANLI-exact	87.2	86.1	86.8	24.8
MANLI-constraint	89.5	86.2	87.8	33.0
Jacana,s2t	91.8	83.4	87.4	25.9
Jacana,t2s	93.7	84.0	88.6	35.3
Jacana, s2t∩t2s	95.4	80.8	87.5	31.3
Jacana, s2t∪t2s	90.3	86.6	88.4	29.6
this-work, s2t	96.6	91.2	93.8	58.6
this-work, t2s	94.3	92.0	93.1	57.0
this-work, s2t∩t2s	89.8	93.3	91.5	46.1
this-work, s2t∪t2s	96.1	88.4	92.1	47.0

From table 1 it is clear that our aligner produces better results than Stanford RTE, Manli, Manli-approx, Manli-exact, Manli-constraint in every parameter i.e P%, R%, F%, E%. If we compare Jacana s2t and our work s2t then also our results are better in every aspect. Similar is for our work t2s, s2t union t2s. But if we compare jacana work s2t intersection t2s then jacana gives better result in Precision but our aligner gives better recall, F, exact match rate. So overall our aligner is giving an increase in comparison parameters. Using Distributional Similarity with other features in jacana led to a broader coverage of words and also helped to align those words which were not aligned previously by previous aligners and hence an overall increase was measured. Values for s2t is higher than t2s because of many-to-one property. Suppose source sentence: Ram is the brother of Shyam. Ram is very intelligent. target sentence: Ram is very intelligent.

Here when we run aligner from s2t then it aligns both source Ram to target Ram. But when we run

aligner from t2s then it aligns target Ram to source 1st Ram but 2nd Ram is the correct answer of alignment. This is the reason due to which results of t2s is lower than s2t. These results shows the value of adding the additional features in the proposed model. The results are compared with each addition.

5 Future Work

The proposed aligner is aligning one pair per second. This speed is slow for applications which use alignment again and again. So it calls for future work on improving the speed of our aligner. This can be done by optimizing our distributional similarity feature algorithm by using some parallel constructs.

6 Conclusion

We introduced this aligner which used only WordNet and POS TAG as lexical resources and provided good results in terms of precision, recall, exact match,f-measure. We compared this aligner with Stanford RTE Aligner, Manli Aligner, Jacana-Token Based Aligner and results showed that our aligner is better if speed is not an issue. In terms of speed, it's doing 1 alignment per second. Recall has increased significantly because of the broader coverage of words by distributional similarity feature. Researchers can take up this aligner and can do even more to improve this aligner's performance.

References

1. **Blunsom, P. & Cohn., T. (2006).** Discriminative word alignment with conditional random fields. *Proceedings of ACL*, ACL, pp. 65–72.
2. **Chambers, N., Cer, D., Grenager, T., Hall, D., Kiddon, C., MacCartney, B., de Marneffe, M. C., Ramage, D., Yeh, E., & Manning, C. D. (2007).** Learning alignments and leveraging natural logic. *Proceedings of the ACL-07 Workshop on Textual Entailment and Paraphrasing*, ACL, pp. 10–20.
3. **De Marneffe, M.-C. & Manning, C. (2008).** The stanford typed dependencies representation. *Coling: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, COLING, pp. 1–8.
4. **Gimpel, K. & Smith, N. A. (2010).** Soft max margin CRFs: training log-linear models with cost functions. *NAACL*, ACL, pp. 733–736.
5. **Kumar Sharma, L. & Mittal, N. (2017).** Prominent feature extraction for evidence gathering in question answering. *Journal of Intelligent and Fuzzy Systems*, IOS Press, pp. 2923–2932.
6. **Lafferty, J. D., McCallum, A., & Pereira, F. C. N. (2001).** Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML, pp. 282–289.
7. **MacCartney, B., Galley, M., & Manning, C. D. (2008).** A phrase-based alignment model for natural language inference. *Proceedings of EMNLP*, EMNLP, pp. 802–811.
8. **MacCartney, B. & Manning, C. (2008).** Modeling semantic containment and exclusion in natural language inference. *Proceedings of ACL*, ACL, pp. 521–528.
9. **McCallum, A., Bellare, K., & Pereira, F. (2005).** A conditional random field for discriminatively - trained finite-state string edit distance. *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, UAI, pp. 521–528.
10. **Thadani, K. & McKeown, K. (2011).** Optimal and syntactically-informed decoding for monolingual phrase-based alignment. *Proceedings of ACL*, ACL, pp. 10–20.
11. **Wan, S., Dras, M., Dale, R., & Paris, C. (2006).** Using dependency-based features to take the para-farce out of paraphrase. *Proceedings of the Australasian Language Technology Workshop*, ALT, pp. 521–528.
12. **Yao, X., Van Durme, B., CallisonBurch, C., & Clark, P. (2013).** A lightweight and high performance monolingual word aligner. *Proceedings of ACL*, ACL, pp. 10–20.

Article received on 06/12/2017; accepted on 15/02/2018.
Corresponding author is Lokesh Kumar Sharma.