Evaluating Correctness of Propositions Using the Web

Mehdi Samadi and **Manuela Veloso** and **Manuel Blum**
Carnegie Mellon University
{msamadi,veloso,mblum}@cs.cmu.edu

Abstract

In this work, we contribute a method that takes advantage of the powerful corpus of the Web data to automatically evaluate the truth of propositions that are stated as multi-argument instantiated predicates, e.g., *City_In_Country (Beijing,China)*. Our approach, OpenEval, automatically converts a given instantiated predicate into a Web search query, then extracts a corresponding set of features from the web pages returned. Initially, OpenEval trains a classifier on a list of predicates by using a set of seed positive examples for each predicate. Each such set furthermore provides negative examples for the other predicates. To evaluate a new query, OpenEval again converts the query into a corresponding set of features extracted from the Web. The extracted features are then used as input to the learned classifier. The classifier output is used to calculate the correctness probability of the input predicate. We experimentally show that OpenEval is significantly superior to the previous related techniques, in particular the Pointwise Mutual Information (PMI) and Never-Ending Language Learner (NELL).

1 Introduction

"Is yogurt healthy?", "Is zero a number?", "Is pumpkin a fruit?", these are a few examples of the most popular questions that have been searched in Google.[1] These questions can be converted to propositions that require assessment of their correct value. In general, evaluating correctness of a proposition is of interest to information processing applications, such as automated Question Answering (QA). Most of the questions and their candidate answers can be converted to a proposition that requires assessment. For example the question "where is Beijing located?" and the corresponding answer "China" can be converted to a proposition "Beijing is located in China". It has been shown that the accuracy of a QA system can be improved by using automated answer validation techniques [4] that assess the correctness of the answers found by the QA system.

Evaluating correctness of a proposition can also be used in Information Extraction (IE) systems to assess the probability that information is extracted correctly [3]. IE systems such as KnowItAll [3], TextRunner [2] and NELL [1] extract a large set of facts from their input text corpus. Etzioni et al. showed [3] that the accuracy of IE systems can be improved by validating the correctness of the extracted facts. In particular, they used search engine hit counts to assign the probability of correctness to the facts that are extracted by their KnowItAll system.

This paper presents *OpenEval*, a new IE approach that is able to **learn** how to **evaluate** correctness of a proposition. OpenEval uses multi-argument predicates such as *City_In_ Country(Beijing,China)* ['Is Beijing a city in China?'] to represent the extracted information. A predicate such as $p(x_1, ..., x_n)$ defines a relationship between entities $(x_1, ..., x_n)$. We call $(x_1, ..., x_n)$ an *instance* of predicate p. As part of training, OpenEval is provided a set of predicates P (e.g., *City(x)*, *City_ In_Country(x,y)*) and a few seed examples for each predicate $p \in P$ (e.g., *Los Angeles* and *New York* for predicate *City*). For each predicate $p \in P$ and i_p, an instance of p, OpenEval first automatically converts i_p into a Web search query and then extracts the corresponding set of features from the Webpages returned by the search engine. Each of the extracted features consists of a single word or a combination of a set of words (e.g., *located-in*). The extracted features are then used to train a classifier (e.g., SVM). The goal of the classifier is to classify each set of the input features to one of the predicates in P.

To validate correctness of an input predicate instance, OpenEval converts the input predicate instance to a search query and extracts a corresponding set of features from the Web. The extracted features are then used as input to the learned classifier. The classifier output is used to calculate the correctness probability of the input predicate.

The baseline for our work is the Pointwise Mutual Information (PMI) technique, that has been widely used in IE [5; 6; 3], and Never-Ending Language Learner(NELL) [1]. We train OpenEval on a set of mutually exclusive predicates that are chosen randomly from 400 predicates in NELL's ontology. We experimentally show that OpenEval improves F1 measure on the test data at least by a factor of two compared to PMI and NELL techniques.

2 Related Work

Researchers have worked on the problem of evaluating correctness of a proposition from both information extraction and question answering points of view. Turney [6] presented an unsupervised technique that uses search engine hit counts to measure the similarity of pairs of words. They also used the PMI technique to measure semantic similarity between two words. Other researchers have used PMI to validate information extraction [5] or to validate a candidate answer in a question answering system [4]. Soderland et al. [5] showed different methods of using PMI which can be used to assign a correctness probability to the extracted facts.

The performance of all the above techniques depends on the accuracy of the search engine hit counts. However, the search engine hit counts are only a crude estimate of the number of matching documents and it has been shown that hit count estimates are not accurate [7]. However, our approach uses the content of the webpages

[1] Similar popular questions can be obtained by entering words, such as "Is" or "Does" in Google, and seeing Google's suggestions.

New York is a vibrant and eclectic place. There is an incredibly diverse range of things to see and do- but there are a few attractions like no other. No visit to **New York** City is complete without visiting the Empire State Building, the **Statue of Liberty** and the Ellis Island Immigration Museum. Ellis Island is where it all began. You will receive a free ferry ride to the Island and the **Statue of Liberty** with the **New York** Pass.

Figure 1: An example of the text that is found for predicate *City_Attraction(Statue of Liberty, New York)*.

that are returned by the search engines for the evaluation and does not rely on the search hit counts.

NELL (Never-Ending Language Learning) [1] is a semi-supervised learning that has been developed as part of the Read the Web (RTW) project at Carnegie Mellon University. It extracts structured information from the unstructured web pages. The input of NELL is an initial ontology (e.g. hundreds of predicates with one or two arguments) and 10 to 15 seed examples for each predicate that is defined in its ontology. Given this input, NELL extracts new instances of predicates from a collection of 500 million webpages. Given an instance of predicate p, we can simply validate it by checking if the input instance is in the NELL's Knowledge Base (KB). In our experimental result, we compare result of OpenEval to NELL.

3 OpenEval

OpenEval considers a set P of predicates p with optional associated keywords K_p for each p, and instantiation i_p for each predicate. For example, for $p =$ *City*, values of i_p are *New York* and *Beijing*, and for $p =$ *City_Country*, values of i_p are (*New York, USA*) and (*Beijing, China)*. OpenEval evaluates the correctness of a combined tuple $t = \langle p, i_p \rangle$, using the keywords K_p, if any.

We describe next in detail the components of OpenEval. Throughout the presentation, we exemplify with a set of predicates $P = \{$*City_Attraction, Country_Currency*$\}$ and a set of corresponding instantiations as two subsets of seed examples of attractions of cities and currencies of varied countries. We further use a set of keywords $K_{ca} = \{$*popular, unique*$\}$ for *City_Attraction*, and $K_{cc} = \emptyset$ (empty set) for *Country_Currency*.

3.1 Feature Extractor

The input to the *feature extractor* is the tuple $t = \langle p, i_p, K_p, N \rangle$, where $p \in P$ is the input predicate, i_p is an instance of predicate p, $K_p \in K$ is a set of optional keywords, and N is the number of webpages that the feature extractor should extract feature from. The output of the feature extractor is a set of *bag-of-features*, denoted by B_{i_p}. Each $b \in B_{i_p}$ is a bag-of-features that is extracted for the input instance i_p. Each *feature* is a single word or a combination of a set of words (e.g., *located-in*). A *bag-of-features* contains a set of features (there may be multiple instances of a feature) that are extracted by the feature extractor. For example, $\{$*must-See, maps, nearby-restaurants, Admission, tickets, prices, entry*$\}$ is part of one of the bag-of-features that is extracted for the predicate *City_Attraction*. In our explanation, B_{i_p} indicates a set of bag-of-features that are extracted for instance $i_p \in I_p$.

Algorithm 1 shows the feature extraction procedure. Given the input tuple $t =< p, i_p, K_p, N >$, the feature extractor first builds the query Q (Lines 3-4). The query Q is built from the name of the predicate p, arguments of the input instance i_p, and the keywords K_p. For example, suppose that *(Statue of Liberty , New York)* is given as an instance of the predicate *City_Attraction*. The query Q would then be $\{$*"Statue of Liberty" "New York" popular unique*$\}$. The feature extractor then searches the query Q in Google and downloads the first N webpages (Line 5). For each webpage W_i that is found by Google, the feature extractor searches the content of W_i and extracts all the features in W_i. The feature extractor finds all the positions p_j in W_i whose words in $pArgs$ appear "close" to each other (Line 8). "Close" means we allow words in $pArgs$ to appear in any arbitrary order and up to a maximum of 10 words can be in between, before, and after them. For each position p_j, consider $\bar{T}_j$ as the text that occurs around $pArgs$. The bag-of-features b is then built from all the words and bigrams in $\bar{T}_j$. All the stop words and also the bigrams that contain two stop words are deleted from b. All the bag-of-features b that are found in webpage W_i are added to B_{W_i} (Line 11). Figure 3.1 shows an example of the text that is found in one of the returned webpages for query Q. The arguments of the predicate *City_Attraction* are shown in bold. Only in two places a maximum of 10 words occur between words in $pArgs$. The underlined text shows the value of $\bar{T}_j$ for each occurrence of words in $pArgs$. The feature extractor returns two bag-of-features in this case: $b_1 = \{$*attractions, few-attractions, attractions-like, visit, No-visit, visit-to, ...* $\}$, and $b_2 = \{$*receive, receive-a, free, a-free, free-berry, berry, ...*$\}$.

Algorithm 1 Feature Extractor

Input: $t =< p, i_p, K_p, N >$←Input Tuple
1: **Function: FeatureExtractor** ($T =< p, i_p, K_p, N >$)
2: $B_{i_p} \leftarrow \phi$ //B_{i_p} is the set of all the bag-of-features for i_p
3: $pArgs \leftarrow$ Arguments of i_p separated by space
4: $Q \leftarrow$ "p $pArgs$ K_p //Q is the search query
5: $W \leftarrow$ Get the first N results of Google for query Q
6: $B_{W_i} \leftarrow \phi$ //B_{W_i} is the list of all the bag-of-features that are extracted from webpage W_i
7: **for all** $Webpages\ W_i \in W$ **do**
8: **for all** occurrences p_j of $pArgs$ in W_i **do**
9: $\bar{T}_j \leftarrow$ extract text around $pArgs$
10: $b \leftarrow$ all the words and bigrams in $\bar{T}_j$
11: Add b to B_{W_i}
12: **end for**
13: Merge all the bag-of-features in B_{W_i} and add to B_{i_p}
14: **end for**
15: **return** B_{i_p}

The final step of the algorithm is to build the set of bag-of-features B_{i_p} from each element of B_{W_i} (Lines 13). To do this, all the features that are extracted from the same webpage W_i, are merged to a single bag-of-features. In this case, only one single bag-of-features is built for each $W_i \in W$. The extracted bag-of-features contains all the words that appear around words in $pArgs$ at different positions of the webpage W_i. For example, b_1 and b_2 are merged to a single bag-of-features: $v = \{$*attractions, few-attractions, attractions-like, visit, No-visit, visit-to, receive, receive-a, free, a-free, free-berry, berry, ...*$\}$.

3.2 Training Data Constructor

The role of the *training data constructor* is to build a set of training data that are later used to train a classifier. The algorithm takes the tuple $T =< P, I, K, N >$ as an input. For each predicate $p \in P$ and each instance $i_p \in I_p$, the feature extractor is called by given the tuple $< p, i_p, K_p, N >$ as its input. The feature extractor extracts and returns a set of bag-of-features B_{i_p} that are extracted for i_p. Each of the bag-of-features $b \in B_{i_p}$ is used as one of the training examples. The label for b is the same as the name of predicate p.

3.3 Classifier

The next step is to train a *classifier* on a set of bag-of-features B that is built by the *training data constructor*. First, each of the bag-of-features $b \in B$ is transformed into a *feature vector* f, where each element corresponds to the frequency of a distinct feature in b. The dimension of the vector is equal to the total number of distinct features (i.e., words and bigrams) that exist in B. The label for the feature vector f is the same as the label of the input bag-of-features b. We train a classifier such as SVM to classify the constructed feature vectors to different classes. Each class represents a predicate $p \in P$.

The trained classifier is later used in the evaluation part of OpenEval to classify the input bag-of-features b_{i_p} to one of the predicates $p \in P$. In this case, b_{i_p} is first transformed to a feature vector and then is given to the trained classifier. The classifier classifies the input feature vector to one of the predicates $p \in P$.

3.4 Evaluator

The input of the *evaluator* is a tuple $t =< p, i_p, K_p, N >$, where p is a predicate $p \in P$, i_p is a candidate instance of predicate p that the evaluator is required to make an assessment, K_p is a set of keywords for predicate p, and N is the number of webpages that evaluator should process. The input keywords K_p should be the same as the set of keywords that are used in the training. The evaluator first calls the feature extractor to extract a set of bag-of-features for the input tuple t. The feature extractor returns B_{i_p} which contains a set of bag-of-features related to the predicate i_p. For each $b_{i_p} \in B_{i_p}$, the trained classifier is then called to classify b_{i_p} to one of the predicates in P.

The correctness probability of the input predicate instance i_p is calculated by dividing the number of times that $b \in B_{i_p}$ is classified as p to the size of B_{i_p}. For example, suppose that *City_Attraction* and *(Statue of Liberty, New York)* are given as p and i_p. Also suppose that feature extractor extracts features from 8 webpages (i.e., $N = 8$). In this case, B_{i_p} contains the maximum of eight bag-of-features, each of which is a set of features that are extracted from the same website. Each $b \in B_{i_p}$ is given to the classifier that is trained in the classification part. The classifier classifies b to one of the predicates $p \in P$. The correctness probability is obtained by the number of times that b is classified as *City_Attraction* divided by the total number of bag-of-features in B_{i_p} (maximum is eight).

To decide if i_p is an instance of predicate p, one can check if the correctness probability that is found by the evaluator is greater than a predefined threshold. Given a set of seed examples as an input, the process of choosing a threshold Tr_p for each predicate p is as follows. First, for each predicate $p \in P$, the seed examples are split into two sets, T_1 and T_2. OpenEval is then trained using all the seed examples in T_1. Now in order to set the value of Tr_p, we calculate the F1 score for all integers between 0 and 100 using the seed examples in set T_2, and the integer which achieves the best F1 score is assigned to Tr_p.

4 Experimental Evaluation

4.1 Setup

OpenEval is tested on two sets of predicates, one set contains predicates with one argument (i.e., categories) and the other contains predicates with two arguments (i.e., relations) that are chosen randomly from 400 predicates of NELL's knowledge base. [2] Our approach is general and can be used for predicates with any number of arguments. However to be able to compare our results with other related work, we have tested OpenEval only on the predicates with one and two arguments. Table 1 lists all of the categories in the leftmost column, and Table 2 lists all the relations in the leftmost column. For each predicate, 15-25 positive seed examples are provided as an input to the OpenEval. OpenEval is trained and tested separately on each set of the predicates with one and two arguments. We have used Support Vector Machines (SVM) as the classification technique in OpenEval.

To evaluate OpenEval, 30 random positive instances are chosen for each of the input predicates from *Wikipedia.org*. Wikipedia contains a set of webpages that have listed instances of different categories and relations (e.g., `http://en.wikipedia.org/wiki/List_of_academic_disciplines`). Each positive test example for one of the predicates is considered as a negative test example for the other predicates since all the predicates are mutually exclusive.

We have compared our technique to both NELL and PMI using standard performance metrics of precision, recall, and F1. NELL extracts instances of categories and relations that are defined in its ontology from a large Web corpus that contains 500 million webpages (the ClueWeb09 corpus) [1]. Given an instance i_p of predicate p, we can simply use NELL to check if i_p is an instance of p. This can be done by checking if i_p exists in the NELL's Knowledge Base (KB) as an instance of predicate p. We have used NELL's KB that is built from 196 iterations. Another baseline for our comparison is PMI technique. Given a predicate p and a predicate instance i_p, the PMI score is calculated as described in [6]. The thresholds for both PMI and OpenEval techniques are calculated using the the same technique that is explained in Section 3.4.

4.2 Experimental Results

Tables 1 and 2 show the Precision (Pr), Recall (Re) and F1 score (F1) of PMI, NELL, and OpenEval for predicates with one argument (i.e., categories) and predicates with two arguments (i.e., relations). The predicates are randomly chosen from NELL's ontology. The experiments are obtained when OpenEval uses 56 webpages (i.e., $N = 56$) for the training and 8 webpages (i.e., $N = 8$) for the evaluation. The first column shows the predicates that are used in the experiments. Across all the predicates, we can see that OpenEval has achieved the best average F1 score (48% for categories and 45% for relations).

Comparing the result of NELL and OpenEval, it can be seen that performance of both NELL and OpenEval are almost the same for the categories. NELL and OpenEval have achieved average F1 score of 43% and 48% while the F1 score of PMI is 13%. NELL's accuracy for relations is significantly less compared to categories. Carlson et al. [1] have also stated that NELL's performance is significantly better for categories compared to relations. Comparing the average F1 score of OpenEval, NELL, and PMI on relations, it can be seen that accuracy of OpenEval is about 2.5 times higher than both NELL and PMI.

5 Conclusion

This paper introduced OpenEval, a novel information extraction approach that is able to automatically evaluate correctness of a predicate instance. OpenEval can be used to validate information that extracted by IE systems or to validate a candidate answer that is extracted by a QA system. We described two main components of OpenEval in detail: *learning* and *evaluation*, and experimentally showed that OpenEval massively outperforms the PMI technique. We also compared the result of our system to NELL's KB. Comparison of the average F1 score of NELL and OpenEval shows that OpenEval outperforms NELL on predicates with two arguments at least by a factor of six. Both NELL and OpenEval have achieved

[2] The predicates are chosen randomly under the condition that they should be mutually exclusive.

Predicate Name	PMI			NELL			OpenEval		
	Pr(%)	Re(%)	F1(%)	Pr(%)	Re(%)	F1(%)	Pr(%)	Re(%)	F1(%)
Academic Field	14	37	21	100	20	33	26	47	33
Animal	5	100	9	100	77	87	22	57	32
Board Game	5	100	9	86	20	32	49	60	54
Emotion	12	70	20	97	93	95	19	67	29
Furniture	8	13	10	100	60	75	38	50	43
Geometric Shape	5	100	9	100	17	29	63	50	56
Sports Equipment	31	13	19	100	5	9	43	20	27
Ethnic Group	10	13	12	100	17	29	23	73	35
ML Algorithm	14	13	14	100	30	46	33	33	33
Religion	5	100	9	100	27	42	66	46	54
Protein	100	13	24	93	47	62	85	77	81
Programming Language	5	100	9	100	5	9	16	57	25
Musician	71	17	27	100	13	24	61	73	67
Scientist	5	100	9	100	5	9	29	63	40
Park	5	100	9	100	27	42	36	67	47
Mountain	5	100	9	100	47	64	53	60	56
Food	5	100	9	100	7	13	62	77	69
Film Festival	5	100	9	100	40	57	69	97	81
Building Material	12	17	14	100	5	9	33	50	39
Disease	67	7	12	100	50	67	50	60	55
Model	5	100	9	100	50	67	57	67	62
Average	**19**	**63**	**13**	**99**	**31**	**43**	**44**	**60**	**48**

Table 1: The performance (F1, precision, recall) of PMI, NELL, and OpenEval techniques for predicates with one argument.

Predicate Name	PMI			NELL			OpenEval		
	Pr(%)	Re(%)	F1(%)	Pr(%)	Re(%)	F1(%)	Pr(%)	Re(%)	F1(%)
Language of Country	31	13	19	100	33	50	29	53	38
Musician Plays Instrument	7	80	13	100	3	6	39	30	34
Company's Office In City	6	93	11	0	0	0	13	13	13
Country Currency	21	23	22	0	0	0	70	70	70
Team Plays Sport	10	83	17	100	40	57	32	73	44
Company Produces Product	8	53	15	0	0	0	30	50	38
Sport Uses Equipment	9	66	16	0	0	0	46	45	46
Airport In City	6	100	12	100	16	28	71	48	58
Politician US Holds Office	6	100	11	0	0	0	33	40	36
Park In City	6	93	12	66	7	13	63	40	49
Stadium Located In City	6	80	11	100	10	18	38	57	45
Museum In City	6	100	12	100	3	6	75	50	60
Aquarium In City	7	83	13	0	0	0	45	43	44
Person Born In City	7	77	12	0	0	0	45	17	24
Team Plays In League	8	93	15	100	100	100	54	47	50
Body part Contains Body part	7	90	12	0	0	0	58	47	52
Drug Has Side Effect	6	100	11	100	13	23	64	53	58
Average	**9**	**78**	**14**	**51**	**13**	**18**	**47**	**46**	**45**

Table 2: The performance (F1, precision, recall) of PMI, NELL, and OpenEval techniques for predicates with two arguments.

almost the same results on predicates with one argument. OpenEval is a general technique, easy to implement, and has relatively low training time.

References

[1] Andrew Carlson, Justin Betteridge, Richard C. Wang, Estevam R. Hruschka Jr., and Tom M. Mitchell. Coupled semi-supervised learning for information extraction. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM 2010)*, 2010.

[2] Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S. Weld. Open information extraction from the web. *Commun. ACM*, 51:68–74, December 2008.

[3] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in knowitall: (preliminary results). In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 100–110, New York, NY, USA, 2004. ACM.

[4] Bernardo Magnini, Matteo Negri, Roberto Prevete, and Hristo Tanev. Is it the right answer? exploiting web redundancy for answer validation. In *In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 425–432, 2002.

[5] Stephen Soderl, Oren Etzioni, Tal Shaked, and Daniel S. Weld. The use of web-based statistics to validate information extraction. In *AAAI Workshop on Adaptive Text Extraction and Mining*.

[6] Peter Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl, 2001.

[7] Ahmet Uyar. Investigation of the accuracy of search engine hit counts. *J. Inf. Sci.*, 35:469–480, August 2009.