TextJoiner: On-demand Information Extraction with Multi-Pattern Queries

Chandra Sekhar Bhagavatula, Thanapon Noraset, Doug Downey
Electrical Engineering and Computer Science
Northwestern University
{csb,nor.thanapon}@u.northwestern.edu,ddowney@eecs.northwestern.edu

1 Introduction

Web Information Extraction (WIE) is the task of automatically extracting knowledge from Web content. On-demand WIE systems such as KNOWITNOW [1] and TEXTRUNNER [2] allow users to query the Web for a textual context that indicates a desired relation. For example, the context "$x invented $y" indicates the `Invented(x, y)` relation. The WIE system is tasked with responding in real-time with a list of argument tuples (e.g. `<A.G. Bell, Telephone>`) of the query relation, extracted from the Web.

However, this on-demand WIE approach falls short for many queries, because individual textual contexts face a difficult trade-off between precision and recall. As a simple example, the patterns "the city $x is located in the state of $y" and "$x , $y" both express the relation `CityLocatedInState(x, y)` to some degree, but the first has high precision and low recall, whereas the second has low precision and high recall.

In this paper, we present a new approach to on-demand WIE that improves the precision and recall of relation extraction. Our approach is conceptually simple: we move beyond queries for a single textual context to consider queries that are conjunctions (i.e. inner joins) and disjunctions over *multiple* contexts. For example, the above example could be expressed as an inner join across three queries:

"city of $x" AND "states such as $y" AND "$x , $y"

Each of the three queries is individually high recall, and by restricting results to only those `(x, y)` tuples that appear across *all three queries*, we can achieve high precision as well. The first two queries ensure that $x and $y are of the proper "type" for the desired relation. We implement this approach in an on-demand WIE system, TextJoiner (TJ).

TJ is faced with two challenges. The first challenge is ranking extractions for a given query based on their estimated correctness. While repetition in a large corpus is an indicator of correctness for *individual* relations [1], TJ must combine confidence estimates across multiple distinct relations. The second challenge is to execute pattern-based extraction and joining at interactive speed. Here, we solve both challenges using language models. TJ uses a combination of n-gram and neural network models to retrieve extractions at interactive speed. TJ then ranks the extractions based on their probabilities of occurrence in text, which are estimated by the language models. We show how the language models can infer correct extractions that *do not* explicitly appear in the query contexts in the corpus, but have high distributional similarity to extractions that do. TJ uses a novel approach that leverages both textual and tabular data to estimate distributional similarity with neural network language models.

Experiments with our system over the Wikipedia corpus demonstrate that TJ's novel features allow it to achieve higher precision in on-demand WIE when compared to baseline techniques and TEXTRUNNER, over a variety of different queries. Finally, we make our system publicly available on the Web.[1]

[1] `http://websail-fe.cs.northwestern.edu/textjoiner/`

2 System Description

In this section we describe the components and run-time processing of a query in TJ.

Query Format: A TJ query is a boolean expression in Conjunctive Normal Form. It is a conjunction of *clauses* that are disjunctions of multiple *sub-queries*. Each sub-query consists of three types of tokens: *ground terms*, *variables* and *wildcards*. A *ground term* is either a string literal or an entity (specified as "[<entity name>]"). In our system, we consider each Wikipedia page as an entity. A *variable* is denoted by a variable name prefixed by "$". The only *wildcard* character that TJ supports is "*", which matches any single token. Each sub-query must contain at least one ground term and at least one variable. In the example described in Section 1, "city of $x", "states such as $y" and "$x , $y" are sub-queries of clauses of the input query `q = city of $x AND states such as $y AND $x , $y`. In "city of $x", "city" and "of" are the ground terms and $x is the variable.

Resources: TJ uses a combination of an n-gram language model (M_{ng}) and a neural network based word-vector model (M_{wv}) to return extractions for an input query. M_{ng} enables efficient pattern matching and n-gram probability estimation. M_{wv} maps each entity to an n-dimensional vector embedding (we use $n = 1,000$ in our experiments). The neural network vector embeddings capture latent semantics. In particular, similar entities tend to appear nearby each other in vector space, enabling TJ to perform *set expansion* to obtain new entities of the same type as a set of seed entities by searching nearby the seeds in vector space. Further, for a given relation, pairs of entities exhibiting the relation tend to be separated by a similar vector in the embedding space; we refer to this vector as a *relation vector* [3]. Set expansion and relation vector estimation using M_{wv} are essential for inferring relations that are not explicitly expressed in text.

As described earlier, TJ leverages both textual and tabular data in its models. We generated a data set D_{txt} containing text of all articles in Wikipedia, in which the anchor text of links was replaced with their target entity identifiers.

To utilize data from Wikipedia tables, we make three important assumptions: (i) tables are oriented row-wise, i.e. entities are mentioned in rows and column headers denote a relation between entities in the same row, (ii) entities in the same column are of the same type, and (iii) the first column in a table that has entities is the *subject column* of the table. In our experience, tables on Wikipedia generally adhere to these assumptions.

Based on these assumptions, we created two more datasets: (i) D_{rt} containing relation triples from all tables in Wikipedia and (ii) D_c containing sets of entities that occur in the same column. For example, from the first table on the "List_of_countries_and_capitals_with_currency_and_language" page, we generated strings of the form "`Algeria capital Algiers`" denoting the relation `CapitalOf(Algeria, Algiers)` to create D_{rt} .[2] The set of entities in the `Country` column can be used to create D_c.

M_{ng} is trained on the union of the datasets D_{txt} and D_{rt} using the Berkley LM package by Pauls et. al. [4]. M_{wv} is trained using the word2vec package by Mikolov et. al. [5, 6, 3] on the union of D_{txt} and D_c to generate vector representation of entities in Wikipedia. Training the M_{wv} model on the union of D_{txt} and D_c is a crude way of combining information from text and tables into a distribution based model. Exploring better techniques of combining textual and tabular data is part of future work.

Run-time Processing: Given the query `q`, TJ evaluates its clauses in ascending order of the number of variables they contain. For each sub-query in a clause, TJ uses M_{ng} to find n-grams that match its literal part and assigns variables with values from the matching n-grams. TJ allows variables to only match entities. Variable assignments from a clause are used in subsequent clauses. Iterating through all clauses results in a list of assignments, $\mathcal{A}_{ng}$, in which an assignment consists of one value for each variable in `q`. TJ sorts $\mathcal{A}_{ng}$ in descending order of a score obtained from a scoring function (described later). The score of an assignment indicates TJ's confidence of the assignment being a correct extraction for a given input query. For our example query `q`, let $\mathcal{A}_{ng}$ contain only two tuples: <$x=Chicago; $y=Illinois> and <$x=Austin; $y=Texas>.

TJ then uses M_{wv} to infer new variable assignments such that they represent the same relation as assignments in $\mathcal{A}_{ng}$. First, TJ uses distributional-similarity of entities to expand the set of values of each variable in `q`. For our example query `q`, let the new set of values for $x be the set $\mathcal{X}$ ={`Denver`, `Los Angeles`} and for $y be the set $\mathcal{Y}$ ={`California`, `Colorado`}.

[2] https://en.wikipedia.org/wiki/List_of_countries_and_capitals_with_currency_and_language

Second, TJ computes the average of the vector offsets between values of pairs of variables in each assignment (e.g. between values of $x and $y). TJ uses the top 20 assignments in $\mathcal{A}_{ng}$ to calculate this vector referred to as the *average relation vector*. Pairs of $x and $y, from $\mathcal{X}$ and $\mathcal{Y}$, whose relation vector is closest to the *average relation vector*, are collected into a new set of assignments, $\mathcal{A}_{wv}$. $\mathcal{A}_{wv}$ is also sorted using the same scoring function that is used to sort $\mathcal{A}_{ng}$. For our example query `q`, $\mathcal{A}_{ng}$ contains the following tuples: `<$x=Denver; $y=Colorado>`, `<$x=Los Angeles; $y=California>`.
Finally, TJ merges $\mathcal{A}_{ng}$ and $\mathcal{A}_{wv}$ preserving the order of assignments and returns the final set of assignments, $\mathcal{A}$ for the input query `q`.

Scoring Functions: To rank assignments, we experiment with three different scoring functions. Each scoring function estimates the correctness of an assignment. Generally, the scoring functions rank an extraction higher when its probability of extraction is larger, normalized by the unigram probability of its arguments. A key challenge is how to combine signals of correctness across multiple queries, and we experiment with three different methods. $\mathcal{S}_L(a)$ assigns a score to an assignment by using only the clause with the most number of distinct variables, as this clause is expected to be the most important. $\mathcal{S}_{All}(a)$ uses all clauses to assign a score for an assignment, multiplying scores across clauses. $\mathcal{S}_R(a)$ also uses all queries, but is based on the assumption that the relative positions of extractions in the clauses are more significant than the raw probability values. $\mathcal{S}_R(a)$ combines probabilities across queries by multiplying ranks, rather than raw probability values as in $\mathcal{S}_{All}(a)$. Formally, for a given assignment $a \in \mathcal{A}$ and query `q` with n clauses, we define:

$$\mathcal{S}_L(a) = s(a, q_n) \tag{1}$$

$$\mathcal{S}_{All}(a) = \prod_{i=1}^{n} s(a, q_i) \tag{2}$$

$$\mathcal{S}_R(a) = \prod_{i=1}^{n} rank(a, q_i) \tag{3}$$

using the following definitions:

$$s(a, q_i) = \sum_{j=1}^{|q_i|} \frac{P(q_i^j \setminus_a)}{\prod_{v \in Var(q_i^j)} P(a^v)}$$

$rank(a, q_i)$: rank of a, when assignments are sorted in ascending order of $s(a, q_i)$

$|q_i|$: number of sub-queries in the clause q_i

q_i^j : refers to the i^{th} clause, j^{th} sub-query in q

$q_i^j \setminus_a$: substitutes variables in q_i^j with their values found in a

a^v : value of variable v in assignment a

$Var(q_i^j)$: set of variables in the sub-query q_i^j

$P(s)$: n-gram probability estimate of the string s from $M_{\mathtt{ng}}$

3 Experiments

We conducted our experiments using a manually curated set of queries. The queries are listed in the first column of Table 2. The corresponding TJ queries can be found on our project webpage.[3]

As described in Section 2, TJ uses data from both text and tables in Wikipedia. We also described how TJ uses a combination of two models, $M_{\mathtt{ng}}$ and $M_{\mathtt{wv}}$, to return extractions for multi-pattern queries. Technically, all features of TJ can be implemented using only an n-gram language model built over Wikipedia text. Thus, to evaluate the usefulness of each additional component of TJ, we measured the performance of three versions of our system - (i) $\text{TJ}_{\text{ng}}^{\text{txt}}$ uses only the n-gram language model built from Wikipedia text. (ii) TJ^{txt} uses both n-gram language model and the word-vector model, but using only Wikipedia text. (iii) TJ is our complete system as described in Section 2.

[3] `http://websail-fe.cs.northwestern.edu/textjoiner/`

Table 1: Evaluating the impact of each component of the system.

	$\mathbf{TJ_{ng}^{txt}}$	$\mathbf{TJ^{txt}}$	**TJ**
`Precision@20`	0.97	0.7	0.78
`Recall'`	0.47	0.56	0.73
`F1'`	0.64	0.63	**0.76**
Percentage of correct extractions that were *inferred*	0%	38%	16%

For this experiment, we chose a subset of 10 queries from our query set and manually labeled the top 20 results returned by each system as true or false indicating an extraction's correctness. Table 1 shows the results. Since it is difficult to define recall for our open ended relations, we adopt a pooled-recall measure (`Recall'`) in which we pool all correct extractions across the three systems and treat them as the set of correct extractions.

Table 1 shows the effectiveness of each component of TJ. In addition to the n-gram model, TJ^{txt} also uses a word-vector model to *infer* facts - i.e. find extractions that do not necessarily occur in the context given in the query. The percentage of such inferred extractions increases from 0% in TJ_{ng}^{txt} to 38% in TJ^{txt} which results in an improvement of recall between TJ_{ng}^{txt} and TJ^{txt}. This shows the effectiveness of using a word-vector model in our system. Moreover, our final system TJ which uses both $M_{\texttt{ng}}$ and $M_{\texttt{wv}}$, and both textual and tabular data, improves the recall further and achieves the highest `F1` score.

Table 2: Comparison of different versions of TJ and TEXTRUNNER. `p` represents precision and "Count" is the total number of assignments returned. For systems for which we do not list counts, the number of results returned is 20 for all queries.

		TJ_R	TJ_L	TJ−1		TEXTRUNNER		$TJ_{All+Gap}$		TJ_{All}	
Query	No. Var	p	p	Count	p	Count	p	Count	p	Time (s)	p
Academy award winners	1	1.00	1.00	20	1.00	20	0.95	20	1.00	5	1.00
Discredited theories	1	0.20	0.40	0	0.00	0	0.00	6	0.50	6	0.50
Drummers who are also singers	1	0.20	0.35	0	0.00	1	0.00	20	0.80	5	0.80
List of musicals	1	0.95	0.95	19	0.95	5	0.80	19	0.95	6	0.95
List of theorems	1	0.55	0.55	3	0.15	20	0.10	7	0.43	6	0.55
Nobel Prize Winners	1	1.00	1.00	20	1.00	20	0.80	4	1.00	6	1.00
Superheroes	1	0.85	0.85	19	0.95	15	0.73	3	1.00	5	0.85
Actors who played villains	2	0.00	0.60	0	0.00	0	0.00	1	1.00	304	0.75
City mayors	2	0.00	1.00	20	1.00	20	1.00	20	1.00	13	1.00
Companies & their product	2	0.95	1.00	0	0.00	5	1.00	20	1.00	12	1.00
Company ceo	2	0.50	0.75	12	0.60	3	1.00	7	1.00	18	0.70
Countries Invasion	2	0.35	0.50	13	0.65	20	1.00	5	1.00	14	0.50
Country capitals	2	0.95	1.00	3	0.15	20	1.00	1	1.00	12	0.95
Discredited theories with their authors	2	0.25	0.40	0	0.00	0	0.00	1	1.00	12	0.40
Drummers & their bands	2	1.00	1.00	20	1.00	0	0.00	20	1.00	11	1.00
Game designer & their games	2	0.70	0.75	0	0.00	4	0.00	13	0.92	11	0.75
City-Country	2	0.10	1.00	9	0.45	20	0.25	2	1.00	44	1.00
Citiy-State	2	1.00	1.00	1	0.05	20	0.00	20	0.80	112	0.80
Countries & their official languages	2	0.95	0.75	14	0.70	20	1.00	20	0.95	21	0.95
Nba stars & their teams	2	0.75	0.80	0	0.00	20	0.85	12	1.00	12	0.90
Novels & authors	2	0.45	0.70	0	0.00	29	0.38	8	1.00	12	0.70
Scientists & their discoveries	2	0.35	0.35	16	0.80	20	0.80	4	1.00	12	0.35
US city mayors	2	1.00	1.00	0	0.00	20	0.80	20	1.00	14	1.00
City-state-country	3	0.40	1.00	0	0.00	0	0.00	20	0.80	215	0.80
MAP@20		0.60	0.78		0.39		0.52		0.93		**0.80**

Table 2 shows the performance of the following systems:

TJ_R: This method uses the scoring function $\mathcal{S}_R(a)$ described in the previous section.

TJ_L: This method uses the scoring function $\mathcal{S}_L(a)$ described in the previous section.

TJ–1: To evaluate the usefulness of enabling joins, we compared TJ with a version of TextJoiner that does not allow joins. Each query is rewritten as a single pattern.

TEXTRUNNER: We also compared TJ with an existing on-demand IE system TEXTRUNNER. While TEXTRUNNER does not allow joins, it does allow constraining variables by Freebase types. We use these constraints when possible.

TJ_{All}: This method uses the scoring function $\mathcal{S}_{All}(a)$ described in the previous section.

$TJ_{All+Gap}$: Queries such as "us state $x" can only have a fixed number of bindings that are factually correct. For such sets, the score of a top ranked binding will be significantly higher than the score of a binding at the bottom. We try a simple approach to predict when the bindings start to become irrelevant. In this system, we cut off the list of results where the drop in the score between consecutive assignments is maximized. This method achieves the highest precision, but the number of results returned drops by 50%. Hence, we selected TJ_{All} as our final version of TJ.

4 Related Work

As discussed in the introduction, our work generalizes the KNOWITNOW system to allow joins across multiple distinct queries. Further, TJ can infer novel extractions using its M_{wv} distributional similarity, and leverages tabular data along with text. In our experiments, each of these components of TJ is shown to improve accuracy.

Our work is closely related to query interfaces for Open Information Extraction. TEXTRUNNER provides a query interface to binary relation tuples extracted from the Web [2]. Cafarella et. al. [7] present a system for structured query access to unstructured text. Unlike these systems, TJ does not commit to any intermediate data model. Instead, it matches user queries directly against the text at run-time, using language models. Our experiments show that our system is more accurate than TEXTRUNNER on our workload.

Sekine's [8] on demand IE system extracts salient relations for a given topic from a given corpus. Unlike that work, TJ allows users to query for a particular relation, over a large corpus. Yin et. al. [9] proposed a system that extracts facts from structured data on the web. By contrast, our focus is on performing joins and incorporating unstructured text data and semi-structured tables. Recently Akbik et. al. [10] introduced an exploratory relation extraction system which allows multi-pattern relation extraction. But, unlike TJ, this system is restricted to binary relations and extraction of facts is performed pre-emptively following the idea of Preemptive Information Extraction [11]. This work also introduces the includes user driven relation extraction, which could be an interesting enhancement for TextJoiner.

5 Conclusion and Future Work

In this paper, we introduced and evaluated the technique of using conjunctions of extraction patterns for WIE. We presented a system, TextJoiner, that can execute pattern-based extractions and joins at run-time. We showed how TextJoiner uses language models to infer facts for a relation even when the facts are not explicitly stated in text. We also presented methods to rank these extractions based on their estimated correctness. We showed that using a combination of language models and combining data in text and tables on the Web improves pattern-based fact extraction. We also made our system publicly available.

In future work, we would like to explore ways to automatically generate query patterns for a relation. A method to convert natural language questions into TJ query format is also desirable. Exploring better ways to combine textual and tabular data is also part of future work. For relations that have a limited number of extractions, automatically filtering out irrelevant extractions is an interesting direction of future work. $TJ_{All+Gap}$ is our simple approach towards solving this problem, but more sophisticated techniques need to be introduced in the future.

Acknolwedgments

This research was supported in part by NSF grant IIS-1351029 and the Allen Institute for Artificial Intelligence.

References

[1] Michael J Cafarella, Doug Downey, Stephen Soderland, and Oren Etzioni. Knowitnow: Fast, scalable information extraction from the web. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 563–570. Association for Computational Linguistics, 2005.

[2] Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, and Mausam Mausam. Open information extraction: The second generation. In *IJCAI*, volume 11, pages 3–10, 2011.

[3] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751. Citeseer, 2013.

[4] Adam Pauls and Dan Klein. Faster and smaller n-gram language models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 258–267. Association for Computational Linguistics, 2011.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[7] Michael J Cafarella, Christopher Re, Dan Suciu, Oren Etzioni, and Michele Banko. Structured querying of web text. In *3rd Biennial Conference on Innovative Data Systems Research (CIDR), Asilomar, California, USA*, 2007.

[8] Satoshi Sekine. On-demand information extraction. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 731–738. Association for Computational Linguistics, 2006.

[9] Xiaoxin Yin, Wenzhao Tan, and Chao Liu. Facto: a fact lookup engine based on web tables. In *Proceedings of the 20th international conference on World wide web*, pages 507–516. ACM, 2011.

[10] Alan Akbik, Thilo Michael, and Christoph Boden. Exploratory relation extraction in large text corpora.

[11] Yusuke Shinyama and Satoshi Sekine. Preemptive information extraction using unrestricted relation discovery. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 304–311. Association for Computational Linguistics, 2006.