Identifying Functional Relations in Web Text

Thomas Lin, Mausam, Oren Etzioni
Turing Center
University of Washington
Seattle, WA 98195, USA
{tlin,mausam,etzioni}@cs.washington.edu

Abstract

Determining whether a textual phrase denotes a functional relation (*i.e.*, a relation that maps each domain element to a unique range element) is useful for numerous NLP tasks such as synonym resolution and contradiction detection. Previous work on this problem has relied on either counting methods or lexico-syntactic patterns. However, determining whether a relation is functional, by analyzing mentions of the relation in a corpus, is challenging due to ambiguity, synonymy, anaphora, and other linguistic phenomena.

We present the LEIBNIZ system that overcomes these challenges by exploiting the synergy between the Web corpus and freely-available knowledge resources such as Freebase. It first computes multiple *typed functionality* scores, representing functionality of the relation phrase when its arguments are constrained to specific types. It then aggregates these scores to predict the global functionality for the phrase. LEIBNIZ outperforms previous work, increasing area under the precision-recall curve from 0.61 to 0.88. We utilize LEIBNIZ to generate the first public repository of automatically-identified functional relations.

1 Introduction

The paradigm of Open Information Extraction (IE) (Banko et al., 2007; Banko and Etzioni, 2008) has scaled extraction technology to the massive set of relations expressed in Web text. However, additional work is needed to better understand these relations, and to place them in richer semantic structures. A step in that direction is identifying the properties of these relations, *e.g.*, symmetry, transitivity and our focus in this paper – functionality. We refer to this problem as *functionality identification*.

A binary relation is functional if, for a given arg1, there is exactly one unique value for arg2. Examples of functional relations are *father, death_date, birth_city*, *etc.* We define a *relation phrase* to be functional if all semantic relations commonly expressed by that phrase are functional. For example, we say that the phrase *'was born in'* denotes a functional relation, because the different semantic relations expressed by the phrase (*e.g.*, *birth_city, birth_year, etc.*) are all functional.

Knowing that a relation is functional is helpful for numerous NLP inference tasks. Previous work has used functionality for the tasks of contradiction detection (Ritter et al., 2008), quantifier scope disambiguation (Srinivasan and Yates, 2009), and synonym resolution (Yates and Etzioni, 2009). It could also aid in other tasks such as ontology generation and information extraction. For example, consider two sentences from a contradiction detection task:
(1) "George Washington *was born in* Virginia." and
(2) "George Washington *was born in* Texas."
As Ritter *et al.* (2008) points out, we can only determine that the two sentences are contradictory if we know that the semantic relation referred to by the phrase *'was born in'* is functional, and that both Virginia and Texas are distinct states.

Automatic functionality identification is essential when dealing with a large number of relations as in Open IE, or in complex domains where expert help

Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pages 1266–1276,
MIT, Massachusetts, USA, 9-11 October 2010.

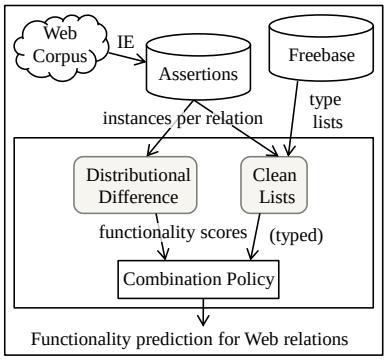

Figure 1: Our system, LEIBNIZ, uses the Web and Freebase to determine functionality of Web relations.

is scarce or expensive (*e.g.*, biomedical texts). This paper tackles automatic functionality identification using Web text. While functionality identification has been utilized as a module in various NLP systems, this is the first paper to focus exclusively on functionality identification as a *bona fide* NLP inference task.

It is natural to identify functions based on triples extracted from text instead of analyzing sentences directly. Thus, as our input, we utilize tuples extracted by TEXTRUNNER (Banko and Etzioni, 2008) when run over a corpus of 500 million webpages. TEXTRUNNER maps sentences to tuples of the form <arg1, *relation phrase*, arg2> and enables our LEIBNIZ system to focus on the problem of deciding whether the relation phrase is a function.

The naive approach, which classifies a relation phrase as non-functional if several arg1s have multiple arg2s in our extraction set, fails due to several reasons: synonymy – a unique entity may be referred by multiple strings, polysemy of both entities and relations – a unique string may refer to multiple entities/relations, metaphorical usage, extraction errors and more. These phenomena conspire to make the functionality determination task inherently statistical and surprisingly challenging.

In addition, a functional relation phrase may appear non-functional until we consider the types of its arguments. In our *'was born in'* example, <George Washington, *was born in*, 1732> does not contradict <George Washington, *was born in*, Virginia> even though we see two distinct arg2s for the same arg1. To solve functionality identification, we need to consider *typed relations* where the relations analyzed are constrained to have specific argument types.

We develop several approaches to overcome these challenges. Our first scheme employs approximate argument merging to overcome the synonymy and anaphora problems. Our second approach, DISTRDIFF, takes a statistical view of the problem and learns a separator for the typical count distributions of functional versus non-functional relations. Finally, our third and most successful scheme, CLEANLISTS, identifies and processes a cleaner subset of the data by intersecting the corpus with entities in a secondary knowledge-base (in our case, Freebase (Metaweb Technologies, 2009)). Utilizing pre-defined types, CLEANLISTS first identifies typed functionality for suitable types for that relation phrase, and then combines them to output a final functionality label. LEIBNIZ, a hybrid of CLEANLISTS and DISTRDIFF, returns state-of-the-art results for our task.

Our work makes the following contributions:

1. We identify several linguistic phenomena that make the problem of corpus-based functionality identification surprisingly difficult.

2. We designed and implemented three novel techniques for identifying functionality based on instance-based counting, distributional differences, and use of external knowledge bases.

3. Our best method, LEIBNIZ, outperforms the existing approaches by wide margins, increasing area under the precision-recall curve from 0.61 to 0.88. It is also capable of distinguishing functionality of typed relation phrases, when the arguments are restricted to specific types.

4. Utilizing LEIBNIZ, we created the first public repository of functional relations.[1]

2 Related Work

There is a recent surge in large knowledge bases constructed by human collaboration such as Freebase (Metaweb Technologies, 2009) and VerbNet (Kipper-Schuler, 2005). VerbNet annotates its verbs with several properties but not functionality. Freebase does annotate some relations with an 'is unique' property, which is similar to functionality, but the number of relations in Freebase is still much

[1]available at `http://www.cs.washington.edu/research/leibniz`

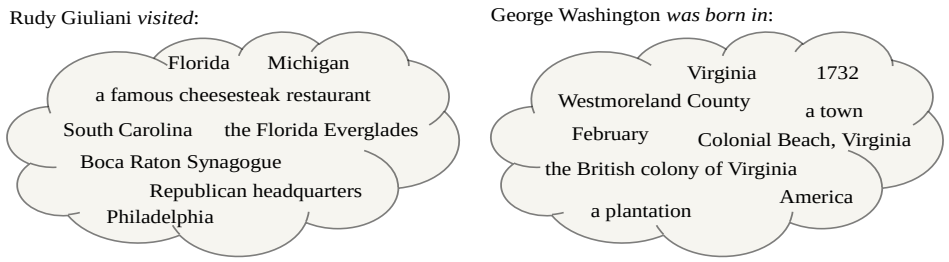

Figure 2: Sample arg2 values for a non-functional relation (*visited*) *vs.* a functional relation (*was born in*) illustrate the challenge in discriminating functionality from Web text.

smaller than the hundreds of thousands of relations existing on the Web, necessitating automatic approaches to functionality identification.

Discovering functional dependencies has been recognized as an important database analysis technique (Huhtala et al., 1999; Yao and Hamilton, 2008), but the database community does not address any of the linguistic phenomena which make this a challenging problem in NLP. Three groups of researchers have studied functionality identification in the context of natural language.

AuContraire (Ritter et al., 2008) is a contradiction detection system that also learns relation functionality. Their approach combines a probabilistic model based on (Downey et al., 2005) with estimates on whether each arg1 is ambiguous. The estimates are used to weight each arg1's contribution to an overall functionality score for each relation. Both argument-ambiguity and relation-functionality are jointly estimated using an EM-like method. While elegant, AuContraire requires substantial hand-engineered knowledge, which limits the scalability of their approach.

Lexico-syntactic patterns: Srinivasan and Yates (2009) disambiguate a quantifier's scope by first making judgments about relation functionality. For functionality, they look for *numeric phrases* following the relation. For example, the presence of the numeric term 'four' in the sentence "the fire *destroyed* four shops" suggests that *destroyed* is not functional, since the same arg1 can destroy multiple things.

The key problem with this approach is that it often assigns different functionality labels for the present tense and past tense phrases of the same semantic relation. For example, it will consider *'lived in'* to be non-functional, but *'lives in'* to be functional, since we rarely say "someone lives in many cities". Since both these phrases refer to the same semantic relation this approach has low precision. Moreover, it performs poorly for relation phrases that naturally expect numbers as the target argument (*e.g.*, *'has an atomic number of'*).

While these lexico-syntactic patterns do not perform as well for our task, they are well-suited for identifying whether a verb phrase can take multiple objects or not. This can be understood as a functionality property of the verb phrase within a sentence, as opposed to functionality of the semantic relation the phrase represents.

WIE: In a preliminary study, Popescu (2007) applies an instance based counting approach, but her relations require manually annotated type restrictions, which makes the approach less scalable.

Finally, functionality is just one property of relations that can be learned from text. A number of other studies (Guarino and Welty, 2004; Volker et al., 2005; Culotta et al., 2006) have examined detecting other relation properties from text and applying them to tasks such as ontology cleaning.

3 Challenges for Functionality Identification

A functional binary relation r is formally defined as one such that $\forall x, y_1, y_2 : r(x, y_1) \wedge r(x, y_2) \Rightarrow y_1 = y_2$. We define a relation string to be functional if all semantic relations commonly expressed by the relation string are individually functional. Thus, under our definition, *'was born in'* and *'died in'* are functional, even though they can take different arg2s for the same arg1, *e.g.*, year, city, state, country, *etc.*

The definition of a functional relation suggests a naive instance-based counting algorithm for identifying functionality. "Look for the number of arg2s for each arg1. If all (or most) arg1s have exactly one arg2, label the relation phrase functional, else, non-functional." Unfortunately, this naive algorithm fails for our task exposing several linguistic phenomena

that make our problem hard (see Figure 2):

Synonymy: Various arg2s for the same arg1 may refer to the same entity. This makes many functional relations seem non-functional. For instance, <George Washington, *was born in*, Virginia> and <George Washington, *was born in*, the British colony of Virginia> are not in conflict. Other examples of synonyms include 'Windy City' and 'Chicago'; '3rd March' and '03/03', *etc.*

Anaphora: An entity can be referred to by using several phrases. For instance, <George Washington, *was born in*, a town> does not conflict with his being born in 'Colonial Beach, Virginia', since 'town' is an anaphora for his city of birth. Other examples include 'The US President' for 'George W. Bush', and 'the superpower' to refer to 'United States'. The effect is similar to that of synonyms – many relations incorrectly appear non-functional.

Argument Ambiguity: <George Washington, *was born in*, 'Kortrijk, Belgium'> in addition to his being born in 'Virginia' suggests that *'was born in'* is non-functional. However, the real cause is that 'George Washington' is ambiguous and refers to different people. This ambiguity gets more pronounced if the person is referred to just by their first (or last name), *e.g.*, 'Clinton' is commonly used to refer to both Hillary and Bill Clinton.

Relation Phrase Ambiguity: A relation phrase can have several senses. For instance '*weighs* 80 kilos' is a different *weighs* than '*weighs* his options'.

Type Restrictions: A closely related problem is type-variations in the argument. *E.g.*, <George Washington, *was born in*, America> *vs.* <George Washington, *born in*, Virginia> both use the same sense of *'was born in'* but refer to different semantic relations – one that takes a country in arg2, and the other that takes a state. Moreover, different argument types may result in different functionality labels. For example, *'published in'* is functional if the arg2 is a year, but non-functional if it is a language, since a book could be published in many languages. We refer to this finer notion of functionality as *typed functionality*.

Data Sparsity: There is limited data for more obscure relations instances and non-functional relation phrases appear functional due to lack of evidence.

Textually Functional Relations: Last but not least, some relations that are not functional may appear functional in text. An example is *'collects'*. We collect many things, but rarely mention it in text. Usually, someone's collection is mentioned in text only when it makes the news. We name such relations *textually functional*. Even though we could build techniques to reduce the impact of other phenomena, no instance based counting scheme could overcome the challenge posed by textually functional relations.

Finally, we note that our functionality predictor operates over tuples generated by an Open IE system. The extractors are not perfect and their errors can also complicate our analysis.

4 Algorithms

To overcome these challenges, we design three algorithms. Our first algorithm, IBC, applies several rules to determine whether two arg2s are equal. Our second algorithm, DISTRDIFF, takes a statistical approach, and tries to learn a discriminator between typical count distributions for functional and non-functional relations. Our final approach, CLEANLISTS, applies counting over a cleaner subset of the corpus, which is generated based on entities present in a secondary KB such as Freebase.

From this section onwards, we gloss over the distinction between a semantic relation and a relation phrase, since our algorithms do not have access to relations and operate only at the phrase level. We use 'relation' to refer to the phrases.

4.1 Instance Based Counting (IBC)

For each relation, IBC computes a global functionality score by aggregating local functionality scores for each arg1. The local functionality for each arg1 computes the fraction of arg2 pairs that refer to the same entity. To operationalize this computation we need to identify which arg2s co-refer. Moreover, we also need to pick an aggregation strategy to combine local functionality scores.

Data Cleaning: Common nouns in arg1s are often anaphoras for other entities. For example, <the company, *was headquartered in*, ...> refers to different companies in different extractions. To combat this, IBC restricts arg1s to proper nouns. Secondly, to counter extraction errors and data bias, it retains

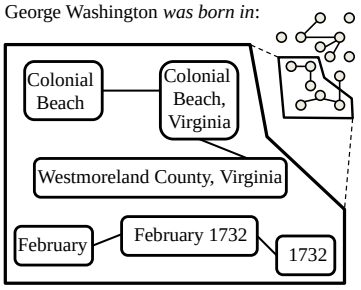

Figure 3: IBC judges that *Colonial Beach* and *Westmoreland County, Virginia* refer to the same entity.

an extraction only once per unique sentence. This reduces the disproportionately large frequencies of some assertions that are generated from a single article published at multiple websites. Similarly, it allows an extraction only once per website url. Moreover, it filters out any arg1 that does not appear at least 10 times with that relation.

Equality Checking: This key component judges if two arg2s refer to the same entity. It first employs weak typing by disallowing equality checks across common nouns, proper nouns, dates and numbers. This mitigates the relation ambiguity problem, since we never compare *'born in*(1732)' and *'born in*(Virginia)'. Within the same category it judges two arg2s to co-refer if they share a content word. It also performs a connected component analysis (Hopcraft and Tarjan, 1973) to take a transitive closure of arg2s judged equal (see Figure 3).

For example, for the relation *'was named after'* and arg1='Bluetooth' our corpus has three arg2s: 'Harald Bluetooth', 'Harald Bluetooth, the King of Denmark' and 'the King of Denmark'. Our equality method judges all three as referring to the same entity. Note that this is a heuristic approach, which could make mistakes. But for an error, there needs to be extractions with the same arg1, relation and similar arg2s. Such cases exist, but are not common. Our equality checking mitigates the problems of anaphora, synonymy as well as some typing.

Aggregation: We try several methods to aggregate local functionality scores for each arg1 into a global score for the relation. These include, a simple average, a weighted average weighted by frequency of each arg1, a weighted average weighted by log of frequency of each arg1, and a Bayesian approach that estimates the probability that a relation is functional using statistics over a small development set. Overall, the log-weighting works the best: it assigns a higher score for popular arguments, but not so high that it drowns out all the other evidence.

4.2 DISTRDIFF

Our second algorithm, DISTRDIFF, takes a purely statistical, discriminative view of the problem. It recognizes that, due to aforementioned reasons, whether a relation is functional or not, there are bound to be several arg1s that look locally functional and several that look locally non-functional. The difference is in the *number* of such arg1s – a functional relation will have more of the former type.

DISTRDIFF studies the count distributions for a small development set of functional relations (and similarly for non-functional) and attempts to build a separator between the two. As an illustration, Figure 4(a) plots the arg2 counts for various arg1s for a functional relation (*'is headquartered in'*). Each curve represents a unique arg1. For an arg1, the x-axis represents the rank (based on frequency) of arg2s and y-axis represents the normalized frequency of the arg2. For example, if an arg1 is found with just one arg2, then x=1 will match with y=1 (the first point has all the mass) and x=2 will match with y=0. If, on the other hand, an arg1 is found with five arg2s, say, appearing ten times each, then the first five x-points will map to 0.2 and the sixth point will map to 0.

We illustrate the same plot for a non-functional relation (*'visited'*) in Figure 4(b). It is evident from the two figures that, as one would expect, curves for most arg1s die early in case of a functional relation, whereas the lower ranked arg2s are more densely populated in case of a non-functional relation.

We aggregate this information using *slope* of the best-fit line for each arg1 curve. For functional relations, the best-fit lines have steep slopes, whereas for non-functional the lines are flatter. We bucket the slopes in integer bins and count the fraction of arg1s appearing in each bin. This lets us aggregate the information into a single slope-distribution for each relation. Bold lines in Figure 4(c) illustrate the average slope-distributions, averaged over ten sample relations of each kind – dashed for non-functional and solid for functional. Most non-functional relations have a much higher probability of arg1s with low magnitude slopes, whereas functional relations are

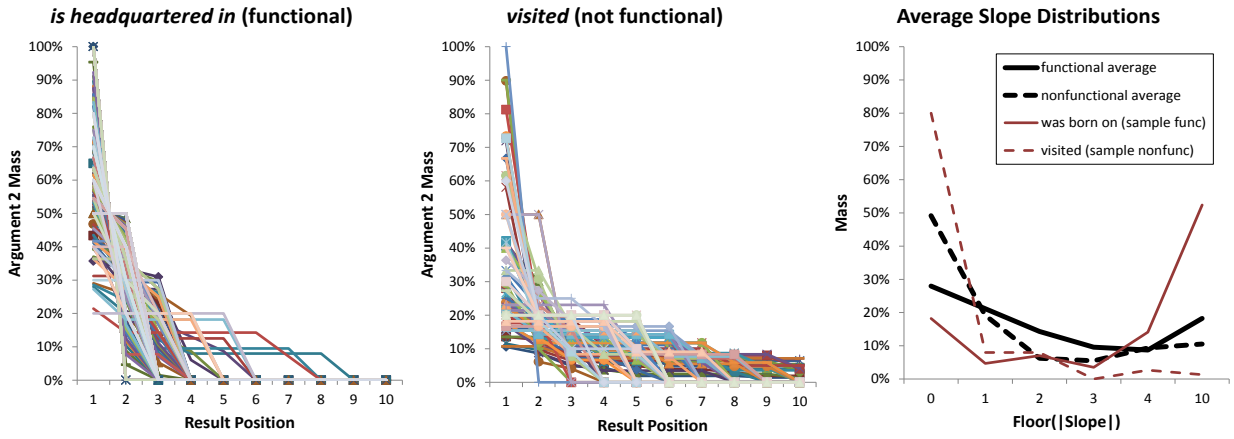

Figure 4: DISTRDIFF: Arg2 count distributions fall more sharply for (a) a sample functional relation, than (b) a sample non-functional relation. (c) The distance of aggregated slope-distributions from average slope-distributions can be used to predict the functionality.

the opposite. Notice that the aggregated curve for *'visited'* in the figure is closer to the average curve for non-functional than to functional and vice-versa for *'was born on'*.

We plot the aggregated slope-distributions for each relation and use the distance from average distributions as a means to predict the functionality. We use KL divergence (Kullback and Leibler, 1951) to compute the distance between two distributions. We score a relation's functionality in three ways using: (1) KLFUNC, its distance from average functional slope-distribution F_{avg}, (2) KLDIFF, its distance from average functional minus its distance from average non-functional N_{avg}, and (3) average of these two scores. For a relation with slope distribution R, the scores are computed as:

$$\text{KLFUNC} = \sum_i R(i) ln \frac{R(i)}{F_{avg}(i)}$$

$$\text{KLDIFF} = \text{KLFUNC} - (\sum_i R(i) ln \frac{R(i)}{N_{avg}(i)})$$

Section 5.2 compares the three scoring functions. A purely statistical approach is resilient to noisy data, and does not need to explicitly account for the various issues we detailed earlier. A disadvantage is that it cannot handle relation ambiguity and type restrictions. Moreover, we may need to relearn the separator if applying DISTRDIFF to a corpus with very different count distributions.

4.3 CLEANLISTS

Our third algorithm, CLEANLISTS, is based on the intuition that for identifying functionality we need not reason over all the data in our corpus; instead, a small but cleaner subset of the data may work best. This clean subset should ideally be free of synonyms, ambiguities and anaphora, and be typed.

Several knowledge-bases such as Wordnet, Wikipedia, and Freebase (Fellbaum, 1998; Wikipedia, 2004; Metaweb Technologies, 2009), are readily and freely available and they all provide clean typed lists of entities. In our experiments CLEANLISTS employs Freebase as a source of clean lists, but we could use any of these or other domain-specific ontologies such as SNOMED (Price and Spackman, 2000) as well.

CLEANLISTS takes the intersection of Freebase entities with our corpus to generate a clean subset for functionality analysis. Freebase currently has over 12 million entities in over 1,000 typed lists. Thus, this intersection retains significant portions of the useful data, and gets rid of most of anaphora and synonymy issues. Moreover, by matching against typed lists, many relation ambiguities are separated as well, since ambiguous relations often take different types in the arguments (*e.g.*, *'ran*(Distance)' *vs. 'ran*(Company)'). To mitigate the effect of argument ambiguity, we additionally get rid of instances in which arg1s match multiple names in the Freebase list of names.

As an example, consider the *'was born in'* relation. CLEANLISTS will remove instances with only 'Clinton' in arg1, since it matches multiple people in Freebase. It will treat the different types, *e.g.*, cities, states, countries, months separately and analyze the functionality for each of these individually.

By intersecting the relation data with argument lists for these types, we will be left with a smaller, but much cleaner, subset of relation data, one for each type. CLEANLISTS analyzes each subset using simple, instance based counting and computes a typed functionality score for each type. Thus, it first computes typed functionality for each relation.

There are two subtleties in applying this algorithm. First, we need to identify the set of types to consider for each relation. Our algorithm currently picks the types that occur most in each relation's observed data. In the future, we could also use a selectional preferences system (Ritter et al., 2010; Kozareva and Hovy, 2010). Note that we remove Freebase types such as *Written Work* from consideration for containing many entities whose primary senses are not that type. For example, both 'Al Gore' and 'William Clinton' are also names of books, but references in text to these are rarely a reference to the written work sense.

Secondly, an argument could belong to multiple Freebase lists. For example, 'California' is both a city and a state. We apply a simple heuristic: if a string appears in multiple lists under consideration, we assign it to the smallest of the lists (the list of cities is much larger than states). This simple heuristic usually assigns an argument to its intended type. On a development set, the error rate of this heuristic is $<4\%$, though it varies a bit depending on the types involved.

CLEANLISTS determines the overall functionality of a relation string by aggregating the scores for each type. It outputs functional if a majority of typed senses for the relation are functional. For example, CLEANLISTS judges *'was born in'* to be functional, since all relevant type restrictions are individually typed functional – everyone is born in exactly one country, city, state, month, *etc.*

CLEANLISTS has a much higher precision due to the intersection with clean lists, though at some cost of recall. The reason for lower recall is that the approach has a bias towards types that are easy to enumerate. It does not have different distances (*e.g.*, 50 kms, 20 miles, *etc.*) in its lists. Moreover, arguments that do not correspond to a noun cannot be handled. For example, in the sentence, "He *weighed* eating a cheeseburger against eating a salad", the arg2 of *'weighed'* can't be matched to a Freebase list. To increase the recall we back off to DISTRDIFF in the cases when CLEANLISTS is unable to make a prediction. This combination gives the best balance of precision and recall for our task. We name our final system LEIBNIZ.

One current limitation is that using only those arg2s that exactly match clean lists leaves out some good data (*e.g.*, a tuple with an arg2 of 'Univ of Wash' will not match against a list of universities that spells it as 'University of Washington'). Because we have access to entity types, using typed equality checkers (Prager et al., 2007) with the clean lists would allow us to recapture much of this useful data. Moreover, the knowledge of functions could apply to building new type nanotheories and reduce considerable manual effort. We wish to study this in the future.

5 Evaluation

In our evaluation, we wish to answer three questions: (1) How do our three approaches, *Instance Based Counting* (IBC), DISTRDIFF, and CLEANLISTS, compare on the functionality identification task? (2) How does our final system, LEIBNIZ, compare against the existing state of the art techniques? (3) How well is LEIBNIZ able to identify typed functionality for different types in the same relation phrase?

5.1 Dataset

For our experiments we test on the set of 887 relations used by Ritter *et al.* (2008) in their experiments. We use the Open IE corpus generated by running TEXTRUNNER on 500 million high quality Webpages (Banko and Etzioni, 2008) as the source of instance data for these relations. Extractor and corpus differences lead to some relations not occurring (or not occurring with sufficient frequency to properly analyze, *i.e.*, ≥ 5 arg1 with ≥ 10 evidence), leaving a dataset of 629 relations on which to test.

Two human experts tagged these relations for functionality. Tagging the functionality of relation phrases can be a bit subjective, as it requires the experts to imagine the various senses of a phrase and judge functionality over all those senses. The inter-annotator agreement between the experts was 95.5%. We limit ourselves to the subset of the data

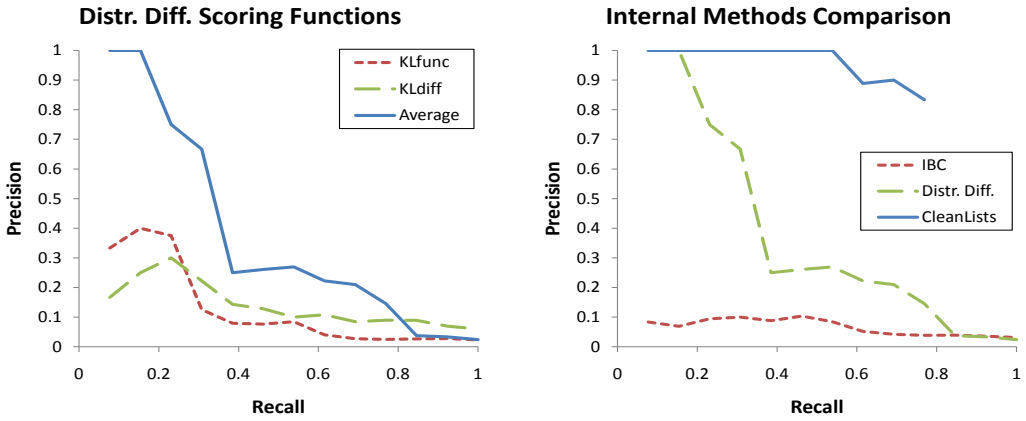

Figure 5: (a) The best scoring method for DISTRDIFF averages KLFUNC and KLDIFF. (b) CLEANLISTS performs significantly better than DISTRDIFF, which performs significantly better than IBC.

on which the two experts agreed (a subset of 601 relation phrases).

5.2 Internal Comparisons

First, we compare the three scoring functions for DISTRDIFF. We vary the score thresholds to generate the different points on the precision-recall curves for each of the three. Figure 5(a) plots these curves. It is evident that the hybrid scoring function, *i.e.*, one which is an average of KLFUNC (distance from average functional) and KLDIFF (distance from average functional minus distance from average non-functional) performs the best. We use this scoring in the further experiments involving DISTRDIFF.

Next, we compare our three algorithms on the dataset. Figure 5(b) reports the results. CLEANLISTS outperforms DISTRDIFF by vast margins, covering a 33.5% additional area under the precision-recall curve. Overall, CLEANLISTS finds the very high precision points, because of its use of clean data. However, it is unable to make 23.1% of the predictions, primarily because the intersection between the corpus and Freebase entities results in very little data for those relations. DISTRDIFF performs better than IBC, due to its statistical nature, but the issues described in Section 3 plague both these systems much more than CLEANLISTS.

To increase the recall LEIBNIZ uses a combination of DISTRDIFF and CLEANLISTS, in which the algorithm backs off to DISTRDIFF if CLEANLISTS is unable to output a prediction.

5.3 External Comparisons

We next compare LEIBNIZ against the existing state of the art approaches. Our competitors are AuContraire and NumericTerms (Ritter et al., 2008; Srinivasan and Yates, 2009). Because we use the AuContraire dataset, we report the results from their best performing system. We reimplement a version of NumericTerms using their list of numeric quantifiers and extraction patterns that best correspond to our relation format. We run our implementation of NumericTerms on a dataset of 100 million English sentences from a crawl of high quality Webpages to generate the functionality labels.

Figure 6(a) reports the results of this experiment. We find that LEIBNIZ outperforms AuContraire by vast margins covering an additional 44% area in the precision-recall curve. AuContraire's AUC is 0.61 whereas LEIBNIZ covers 0.88. A Bootstrap Percentile Test (Keller et al., 2005) on F_1 score found the improvement of our techniques over AuContraire to be statistically significant at $\alpha = 0.05$. NumericTerms does not perform well, because it makes decisions based only on the local evidence in a sentence, and does not integrate the knowledge from different occurrences of the same relation. It returns many false positives, such as *'lives in'*, which appear functional to the lexico-syntactic pattern, but are clearly non-functional, *e.g.*, one could live in many places over a lifetime.

An example of a LEIBNIZ error is the *'represented'* relation. LEIBNIZ classifies this as functional, because it finds several strongly functional senses (*e.g.*, when a person represents a country),

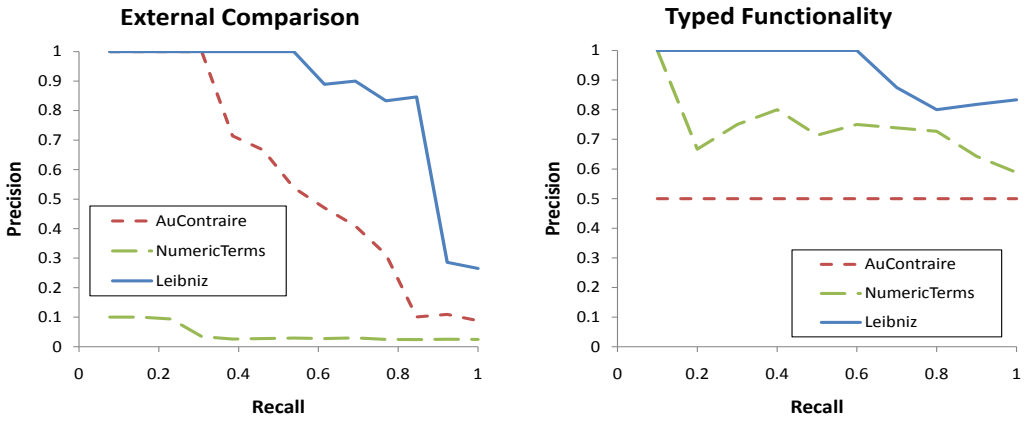

Figure 6: (a) LEIBNIZ, which is a hybrid of CLEANLISTS and DISTRDIFF, achieves 0.88 AUC and outperforms the 0.61 AUC from AuContraire (Ritter et al., 2008) and the 0.05 AUC from NumericTerms (Srinivasan and Yates, 2009). (b) LEIBNIZ is able to tease apart different senses of polysemous relations much better than other systems.

but the human experts might have had some non-functional senses in mind while labeling.

5.4 Typed Functionality

Next, we conduct a study of the typed functionality task. We test on ten common polysemous relations, each having both a functional and a non-functional sense. An example is the *'was published in'* relation. If arg2 is a year it is functional, *e.g.* <Harry Potter 5, *was published in*, 2003>. However, *'was published in*(Language)' is not functional, *e.g.* <Harry Potter 5, *was published in*, [French / Spanish / English]>. Similarly, *'will become*(Company)' is functional because when a company is renamed, it transitions away from the old name exactly once, *e.g.* <Cingular, *will become*, AT&T Wireless>. However, *'will become*(government title)' is not functional, because people can hold different offices in their life, *e.g.*, <Obama, *will become*, [Senator / President]>.

In this experiment, a simple baseline of predicting the same label for the two types of each relation achieves a precision of 0.5. Figure 6(b) presents the results of this study. AuContraire achieves a flat 0.5, since it cannot distinguish between types. NumericTerms can be modified to distinguish between basic types – check the word just after the numeric term to see whether it matches the type name. For example, the modified NumericTerms will search the Web for instances of *"was published in* [numeric term] *years" vs. "was published in* [numeric term] *languages"*. This scheme works better when the type name is simple (*e.g.*, languages) rather than complex (*e.g.*, government titles).

LEIBNIZ performs the best and is able to tease apart the functionality of various types very well. When LEIBNIZ did not work, it was generally because of textual functionality, which is a larger issue for typed functionality than general functionality. Of course, these results are merely suggestive – we perform a larger-scale experiment and generate a repository of typed functions next.

6 A Repository of Functional Relations

We now report on a repository of typed functional relations generated automatically by applying LEIBNIZ to a large collection of relation phrases. Instead of starting with the most frequent relations from TEXTRUNNER, we use OCCAM's relations (Fader et al., 2010) because they are more specific. For instance, where TEXTRUNNER outputs an underspecified tuple, <Gold, *has*, an atomic number of 79>, OCCAM extracts <Gold, *has an atomic number of*, 79>. OCCAM enables LEIBNIZ to identify far more functional relations than TEXTRUNNER.

6.1 Addressing Evidence Sparsity

Scaling up to a large collection of typed relations requires us to consider the size of our data sets. For example, consider which relation is more likely to be functional—a relation with 10 instances all of which indicate functionality versus a relation with 100 instances where 95 behave functionally.

To address this problem, we adapt the likelihood ratio approach from Schoenmackers *et al.* (2010).

For a typed relation with n instances, f of which indicate functionality, the G-test (Dunning, 1993), $G = 2*(f*\ln(f/k)+(n-f)*\ln((n-f)/(n-k)))$, provides a measure for the likelihood that the relation is not functional. Here k denotes the evidence indicating functionality for the case where the relation is not functional. Setting $k = n*0.25$ worked well for us. This G-score replaces our previous metric for scoring functional relations.

6.2 Evaluation of the Repository

In CLEANLISTS a factor that affects the quality of the results is the exact set of lists that is used. If the lists are not clean, results get noisy. For example, Freebase's list of *films* contains 73,000 entries, many of which (*e.g.*, "*Egg*") are not films in their primary senses. Even with heuristics such as assigning terms to their smallest lists and disqualifying dictionary words that occur from large type lists, there is still significant noise left.

Using LEIBNIZ with a set of 35 clean lists on OCCAM's extraction corpus, we generated a repository of 5,520 typed functional relations. To evaluate this resource a human expert tagged a random subset of the top 1,000 relations. Of these relations 22% were either ill-formed or had non-sensical type constraints. From the well-formed typed relations the precision was estimated to be 0.8. About half the errors were due to textual functionality and the rest were LEIBNIZ errors. Some examples of good functions found include *isTheSequelTo(videogame)* and *areTheBirthstoneFor(month*). An example of a textually functional relation found is *wasTheFounderOf(company)*.

This is the first public repository of automatically-identified functional relations. Scaling up our data set forced us to confront new sources of noise including extractor errors, errors due to mismatched types, and errors due to sparse evidence. Still, our initial results are encouraging and we hope that our resource will be valuable as a baseline for future work.

7 Conclusions

Functionality identification is an important subtask for Web-scale information extraction and other machine reading tasks. We study the problem of predicting the functionality of a relation phrase automatically from Web text. We presented three algorithms for this task: (1) instance-based counting, (2) DISTRDIFF, which takes a statistical approach and discriminatively classifies the relations using average arg-distributions, and (3) CLEANLISTS, which performs instance based counting on a subset of clean data generated by intersection of the corpus with a knowledge-base like Freebase.

Our best approach, LEIBNIZ, is a hybrid of DISTRDIFF and CLEANLISTS, and outperforms the existing state-of-the-art approaches by covering 44% more area under the precision-recall curve. We also observe that an important sub-component of identifying a functional relation phrase is identifying typed functionality, *i.e.*, functionality when the arguments of the relation phrase are type-constrained. Because CLEANLISTS is able to use typed lists, it can successfully identify typed functionality.

We run our techniques on a large set of relations to output a first repository of typed functional relations. We release this list for further use by the research community.[2]

Future Work: Functionality is one of the several properties a relation can possess. Others include selectional preferences, transitivity (Schoenmackers et al., 2008), mutual exclusion, symmetry, *etc.* These properties are very useful in increasing our understanding about these Open IE relation strings. We believe that the general principles developed in this work, for example, connecting the Open IE knowledge with an existing knowledge resource, will come in very handy in identifying these other properties.

Acknowledgements

We would like to thank Alan Ritter, Alex Yates and Anthony Fader for access to their data sets. We would like to thank Stephen Soderland, Yoav Artzi, and the anonymous reviewers for helpful comments on previous drafts. This research was supported in part by NSF grant IIS-0803481, ONR grant N00014-08-1-0431, DARPA contract FA8750-09-C-0179, a NDSEG Fellowship, and carried out at the University of Washington's Turing Center.

[2] available at `http://www.cs.washington.edu/research/leibniz`

References

M. Banko and O. Etzioni. 2008. The tradeoffs between open and traditional relation extraction. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*.

M. Banko, M. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. 2007. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*.

A. Culotta, A. McCallum, and J. Betz. 2006. Integrating probabilistic extraction models and data mining to discover relations and patterns in text. In *Proceedings of the HLT-NAACL*.

D. Downey, O. Etzioni, and S. Soderland. 2005. A probabilistic model of redundancy in information extraction. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*.

T. Dunning. 1993. Accurate methods for the statistics of surprise and coincidence. In *Computational Linguistics*, volume 19.

A. Fader, O. Etzioni, and S. Soderland. 2010. Identifying well-specified relations for open information extraction. (In preparation).

C. Fellbaum, editor. 1998. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA.

N. Guarino and C. Welty. 2004. An overview of OntoClean. In *Handbook of Ontologies in Information Systems*, pages 151–172.

J. Hopcraft and R. Tarjan. 1973. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16:372–378.

Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. 1999. TANE: An efficient algorithm for discovering functional and approximate dependencies. In *The Computer Journal*.

M. Keller, S. Bengio, and S.Y. Wong. 2005. Benchmarking non-parametric statistical tests. In *Advances in Neural Information Processing Systems (NIPS) 18*.

K. Kipper-Schuler. 2005. Verbnet: A broad-coverage, comprehensive verb lexicon. In *Ph.D. thesis. University of Pennsylvania*.

Z. Kozareva and E. Hovy. 2010. Learning arguments and supertypes of semantic relations using recursive patterns. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*.

S. Kullback and R.A. Leibler. 1951. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86.

Metaweb Technologies. 2009. Freebase data dumps. In *http://download.freebase.com/datadumps/*.

A-M. Popescu. 2007. Information extraction from unstructured web text. In *Ph.D. thesis. University of Washington*.

J. Prager, S. Luger, and J. Chu-Carroll. 2007. Type nanotheories: a framework for term comparison. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management (CIKM)*.

C. Price and K. Spackman. 2000. SNOMED clincal terms. In *British Journal of Healthcare Computing & Information Management*, volume 17.

A. Ritter, D. Downey, S. Soderland, and O. Etzioni. 2008. It's a contradiction - no, it's not: A case study using functional relations. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

A. Ritter, Mausam, and O. Etzioni. 2010. A latent dirichlet allocation method for selectional preferences. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*.

S. Schoenmackers, O. Etzioni, and D. Weld. 2008. Scaling textual inference to the web. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

S. Schoenmackers, J. Davis, O. Etzioni, and D. Weld. 2010. Learning first-order horn clauses from web text. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

P. Srinivasan and A. Yates. 2009. Quantifier scope disambiguation using extracted pragmatic knowledge: Preliminary results. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

J. Volker, D. Vrandecic, and Y. Sure. 2005. Automatic evaluation of ontologies (AEON). In *Proceedings of the 4th International Semantic Web Conference (ISWC)*.

Wikipedia. 2004. Wikipedia: The free encyclopedia. In *http://www.wikipedia.org*. Wikimedia Foundation.

H. Yao and H. Hamilton. 2008. Mining functional dependencies from data. In *Data Mining and Knowledge Discovery*.

A. Yates and O. Etzioni. 2009. Unsupervised methods for determining object and relation synonyms on the web. In *Journal of Artificial Intelligence Research*, volume 34, pages 255–296.