Explainable Fact Checking with Probabilistic Answer Set Programming

Naser Ahmadi[*]**, Joohyung Lee**[#]**, Paolo Papotti**[*]**, Mohammed Saeed**[*]
[*]EURECOM, France - [#]Arizona State University, USA
{name.surname}@eurecom.fr, joolee@asu.edu

Abstract

One challenge in fact checking is the ability to improve the transparency of the decision. We present a fact checking method that uses reference information in knowledge graphs (KGs) to assess claims and explain its decisions. KGs contain a formal representation of knowledge with semantic descriptions of entities and their relationships. We exploit such rich semantics to produce interpretable explanations for the fact checking output. As information in a KG is inevitably incomplete, we rely on logical rule discovery and on Web text mining to gather the evidence to assess a given claim. Uncertain rules and facts are turned into logical programs, and the checking task is modeled as an inference problem in a probabilistic extension of answer set programs. Experiments show that the probabilistic inference enables the efficient labeling of claims with interpretable explanations, and the quality of the results is higher than state-of-the-art baselines.

1 Introduction

Due to the increase of sources spreading false information, computational *fact checking* has been proposed to support journalists and social media platforms with automatic verification of textual content (Babakar and Moy, 2016). We focus on *claims* that contain factual statements, such as "William Durant was the founder of Chevrolet," and their verification against reference data, i.e., *Knowledge Graphs* (KGs). Assuming entities and relations involved in "worth-checking" claims have been identified (Hassan et al., 2017; Jaradat et al., 2018), KGs are exploited to compute the *veracity* of claims expressed as structured data.

A KG is a structured representation of information which stores real-world entities as nodes, and relationships between them as edges. Entities and relations have semantic descriptions in the form of types and properties associated with them. KGs store large amounts of factual information and several of them are publicly available (Suchanek et al., 2007). For example, the English version of DBpedia stores 6M entities and 9B relation triples.

Given a KG K and a claim f, several approaches have been developed to estimate if f is a valid claim in K. In some of these methods, facts in the KG are leveraged to create features, such as paths (Shi and Weninger, 2016; Ciampaglia et al., 2015) or embeddings (Bordes et al., 2013; Socher et al., 2013), which are then used by classifiers to label as true or false a given test claim. Other methods rely on searching for occurrences of the given claim on Web pages (Dong et al., 2014; Popat et al., 2018). However, such models are based on Machine Learning (ML) classifiers that in the best case can report the source of evidence for a decision but *lack the ability to provide comprehensible descriptions* of how a decision has been taken for a given claim.

To address this problem and effectively support transparent content moderation, we use existing KGs as sources of evidence, together with logical reasoning to make fact checking decisions. The key idea is to assess as true or false a given claim and to provide *human-interpretable explanations* for such decision in the form of supporting and contradicting evidence. Declarative Horn rules defined over the KG, such as those in Table 1, guide the decision process and provide semantic arguments for the conclusion. For example, "William Durant was the founder of Chevrolet" is marked as true and justified by the facts that Durant is a *key person* for Chevrolet and he *founded* another car company. This explanation comes from the first rule in the table[1]. On the other hand, "Elon Musk was the founder of Chevrolet" is marked as false

---

[1]*Product* models the pairs (company **c**, product **p** of **c**).

```
0.75: foundedBy(a,b) ← keyPerson(a,b), foundedBy(c,b), product(c,d), product(a,d).
0.76: foundedBy(a,b) ← distributor(c,b), distributor(c,d), foundedBy(a,d).
0.97: negfoundedBy(a,b) ← foundingYear(a,c), birthYear(b,d), >(d,c).
0.56: negfoundedBy(a,b) ← foundedBy(a,c), relative(d,c), occupation(d,b).
0.67: negfoundedBy(a,b) ← parentCompany(b,c), subsidiary(c,d), parentCompany(d,a).
```

Table 1: Example of discovered rules with their support for predicate *foundedBy* in DBpedia.

with the explanation that Musk was born after the company foundation year (third rule in the table).

Unfortunately, two issues make the generation of such explanations hard. First, in general, KGs do not come with the rich rules we need in our task. To address this issue, we exploit rule mining approaches (Galárraga et al., 2015; Ortona et al., 2018), which automatically learn logical rules for a given KG (e.g., Table 1). Second, KGs have data quality issues due to the automatic methods that are used to build them at scale. Information stored in KGs is inevitably incomplete (Open World Assumption - OWA) and noisy, because of errors coming from the sources and the automatic extractors (Dong et al., 2014). For these reasons, in many cases, rules cannot be triggered. We identify these cases and resort to mining Web pages to get evidence for missing facts that are crucial to reach a decision for a claim (Popat et al., 2018).

Discovering rules and mining facts enable a fully automatic system, but a new challenge arises from these approaches. Both rules and mined facts are *uncertain*, i.e., they come with a (possibly low) measure of the probability of being correct. To address this third challenge, we use probabilistic answer set programming (Lee and Wang, 2016). The reasoner is the enabler of the inference that combines the evidence in producing a fact checking decision with its explanation.

Our main contribution is a fully *automated and interpretable fact checking system* that effectively exploits uncertain evidence (Section 3). Experimental results on a real KG show that our method (i) obtains qualitative results that are comparable or better than existing black-box ML methods and (ii) outputs human consumable explanations (Section 4). We conclude by discussing related work and open problems in Section 5 and Section 6, respectively.

2 Preliminaries

In this section, we describe the main building blocks of our framework and define our problem.

Knowledge Graph. An RDF KG is a database

representing information with triples (or *facts*) $p(s,o)$ where a *predicate p* connects a *subject s* and an *object o*. For example, the fact that E. Musk was born in 1971 is expressed with a triple *birthYear*(E. Musk, 1971). In a triple, the subject is an *entity*, i.e., a real-world concept; the object is either an entity or a *literal*, i.e., primitive types such as number, date, and string; and the triple predicate specifies a relationship between subject and object. We call a triple to be assessed a *claim*.

Rule Mining in KGs. In our framework, we exploit algorithms for mining declarative Horn rules from KGs (Galárraga et al., 2015; Ortona et al., 2018). A Horn rule has the form:

$$h(x,y) \leftarrow B \qquad (1)$$

where $h(x,y)$ is a single atom (head of the rule) and B (body of the rule) is a conjunction of atoms

$$B_1(z_1,z_2) \wedge B_2(z_3,z_4) \wedge \cdots \wedge B_n(z_{2n-1},z_{2n}).$$

An atom is a predicate connecting two variables, two entities, an entity and a variable, or a variable and a constant (string or number). A mining algorithm outputs positive rules (e.g., *spouse* in the head), which identify relationships between entities, e.g., "if two persons have a child in common, they are in the spouse relation", and negative rules (*negspouse* in the head), which identify data contradictions, e.g., "if two persons are in the parent relation, one cannot be the spouse of the other".

A fact is derived from a rule if all the variables in the body of the rule can be replaced with constants in the KG. For example, consider again Table 1 and the negative rule: *negfoundedBy(a,b) ← foundingYear(a,c), birthYear(b,d), >(d,c)*. We can derive *negFoundedBy*(E. Musk, Chevrolet) because there is a replacement for the body of the rule, i.e, "foundingYear(Chevrolet,1911), birthYear(E. Musk,1971), >(1971,1911)".

For rule mining, we adopt RUDIK (Ortona et al., 2018), which, for every KB predicate outputs rules together with a measure of support.

Assessment of Claims on the Web. As KGs are usually incomplete, we also exploit textual docu-

ments for our analysis. Text mining systems get as input a claim *c* expressed in natural language and analyze *c*'s credibility w.r.t. relevant Web documents. The systems exploit the joint interaction among language style of documents, their stance towards a claim, and source trustworthiness.

For example, consider the claim *foundedBy*(Chevrolet, W. Durant), which is not in the KG, and positive rule from Table 1: *foundedBy(a,b) ← keyPerson(a,b), foundedBy(c,b), product(c,d), product(a,d)*. Assume the KG contains the facts *keyPerson*(Chevrolet, W. Durant), *foundedBy*(GM, W. Durant), and *product*(GM, Automobile), but it misses the product information for Chevrolet. It can be a false fact or a true one missing from the KG (OWA). We therefore test *product*(Chevrolet, Automobile) with the text mining system and obtain that, according to Web documents, the fact is true with confidence 0.57.

In our framework, we adopt CREDEYE, a state of the art system for the automatic credibility assessment of textual claims (Popat et al., 2016). To extract Web articles relevant to the input claim, it uses a commercial search engine (i.e., Bing). Each document is divided into a set of overlapping snippets, and snippets that are strongly related to the claim in terms of unigram and bigram are extracted. Snippets are then used to compute *support* and *refute* scores with logistic regression classifiers trained on claims and evidence documents from the Snopes fact checking repository. The scores are fed as features into a classifier with L1-regularization, distantly trained on Snopes.

(Probabilistic) Answer Set Programming. Given a claim, we collect the rules, the evidence (from the KG and the Web sites), and cast fact checking as a reasoning problem. For this task, we adopt LP^{MLN} (Lee and Wang, 2016), a probabilistic extension of answer set programs with the concept of weighted rules from Markov Logic. In ASP, search problems are reduced to computing *stable models* (a.k.a. answer sets), a set of beliefs that are described by the program. In the case of a Horn program, the stable models coincide with the minimal models, but they differ as soon as the program allows more expressive language constructs, such as negation, defaults, and aggregates. We refer to (Gelfond and Lifschitz, 1988; Lee et al., 2008; Ferraris et al., 2011) for the definitions of stable models.

LP^{MLN} extends the (deterministic) stable model

semantics by embracing the concept of weighted rules. In LP^{MLN}, a weight is assigned to each rule so that the more rules a stable model satisfies, the larger weight it gets, and the probability of the stable model is computed by normalizing its weight among all stable models. In our setting, given a set of rules and evidence facts, we want to see if the given claim belongs to the stable model.

More precisely, let σ be a signature as in first-order logic. An LP^{MLN} program Π is a finite set of weighted rules of the form:

$$w : A \leftarrow B \qquad (2)$$

where *A* is a disjunction of atoms of σ, *B* is a conjunction of literals (atoms and negated atoms) of σ, and *w* is a real number or the symbol α. When *A* is $\perp$ (the empty disjunction), the rule asserts that *B* should be false in the stable model. An LP^{MLN} rule (2) is called *soft* if *w* is a real number or *hard* if *w* is α. An LP^{MLN} program is *ground* if its rules contain no variables. An LP^{MLN} program Π that contains variables is identified with a ground LP^{MLN} program $gr_\sigma[\Pi]$ which is obtained from Π by replacing every variable with every ground term of σ. The weight of a ground rule in $gr_\sigma[\Pi]$ is the same as the weight of the corresponding rule in Π. By $\overline{\Pi}$ we denote the unweighted logic program obtained from Π, i.e., $\overline{\Pi} = \{R \mid w : R \in \Pi\}$.

For a ground LP^{MLN} program Π, Π_I denotes the set of rules $w : R$ in Π such that *I* satisfies *R* (denoted $I \models R$) and SM[Π] denotes the set $\{I \mid I$ is a (deterministic) stable model of $\overline{\Pi_I}\}$. The (unnormalized) weight of *I* under Π is defined as:

$$W_\Pi(I) = \begin{cases} exp(\sum_{w:R\in\Pi_I} w) & \text{if } I \in \text{SM}[\Pi]; \\ 0 & \text{otherwise.} \end{cases}$$

The probability of *I* under Π is the normalized weight defined as: $P_\Pi(I) = \lim_{\alpha\to\infty} \frac{W_\Pi(I)}{\sum_{J\in\text{SM}[\Pi]} W_\Pi(J)}$.

LPMLN2ASP (Lee et al., 2017) is an implementation of LP^{MLN} using ASP solver CLINGO. The system returns the most probable stable models. In our problem formulation, given a claim *p*(x,y), we identify the rules that have predicate *p* or *negp* in the conclusion and the evidence facts for the bodies of such rules. We then run LPMLN2ASP and check if *p* or *negp* are in the stable model.

Problem Statement. Given an input claim to be verified and a KG, our goal is to compute an assessment of the veracity of the claim and the
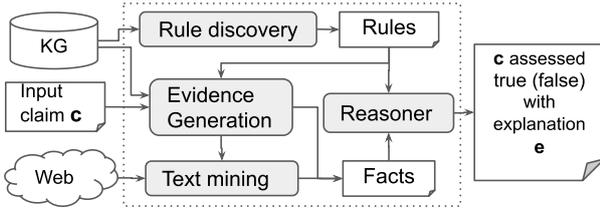
Figure 1: Our Fact checking framework EXPCLAIM.

explanations for such decision, expressed as the union of substitutions for the body of the rules that have triggered the inference in the reasoner.

The uncertainty in the discovered rules and in the facts extracted from the Web make the problem challenging and the role of the reasoner important.

Limits of Existing Solutions. Both the text mining and the rule generation system can be used individually as fact checking tools according to our problem definition. However, they both have strong limitations. The uncertain rules alone cannot make a clear assessment decision in many cases because of (i) conflicting rules both supporting and refusing a fact at the same time, and (ii) lack of evidence in the KG. The Web mining cannot provide semantic explanations and also suffers from the cases where there is no enough evidence to obtain an answer. These limitations also apply for other ML fact checking systems (Ciampaglia et al., 2015; Shi and Weninger, 2016; Bordes et al., 2013; Socher et al., 2013) and motivate our choice to use a unified framework to combine both sources of signals with a probabilistic reasoner.

3 Framework

Figure 1 shows our framework, EXPCLAIM. The *Rule discovery* module takes as input the KG K to generate the rules. We then convert the discovered rules Σ into the input language of the reasoner, where the weight of a rule is its *support*. For the given claim $c : p(x,y)$, $p \in K$ and rules Σ, the *Evidence Generation* module collects relevant evidence facts (triples satisfying the body of the rules) from the KG and from Web with the *Text mining* module. We then feed rules and evidence to the *Reasoner* module, where different modes of computation can be used to infer if $p(x,y)$ or $negp(x,y)$ is in the answer set. The reasoner output includes a human-interpretable explanation for the decision. The details of the main steps are given next.

3.1 Rule Generation

Consider a claim $c : p(x,y)$ with $p \in K$, our first step is to obtain the set of rules Σ.

Rule Discovery: The rule discovery module starts by generating M positive and M negative examples for p. Positive examples are (x,y) entity pairs s.t. $p(x,y) \in K$, and negative examples are (x,y) pairs that satisfy the following conditions (Ortona et al., 2018):

- $p(x,y) \notin K$;
- there is either some $y' \neq y$ s.t. $p(x,y') \in K$ or some $x' \neq x$ s.t. $p(x',y) \in K$;
- there is some $p' \neq p$ s.t. $p'(x,y) \in K$.

RUDIK uses the examples and the KG to mine positive and negative rules (Σ) for p.

Consider the mining of positive rules for predicate *spouse*. Positive examples are pairs of married people, and negative examples are pairs of people who are not married to each other, according to the three conditions above. Given the examples, the algorithm outputs *approximate* rules that (i) maximize the coverage of the positive examples and (ii) minimize the coverage of the negative ones. The example sets switch role for the discovery of negative rules, i.e., not married people play the role of the positive examples.

As in association rule mining, the support s of each rule is computed as the *support value* of the rule divided by the number of examples used in the rule discovery step (Galárraga et al., 2015).

Convert Rules into LP^{MLN}: Rules in Σ are rewritten into the input language of LPMLN2ASP with their weights. For instance, for the *spouse* predicate, a positive rule is rewritten into LP^{MLN} as

$$w : spouse(a,b) \leftarrow child(a,c), parent(c,b). \quad (3)$$

An original support s equal to 0 corresponds to a weight w of $-\infty$ and a support of 1 to a weight of $+\infty$. We convert the rule support into a weight for a program with the equation: $w = \ln \frac{s}{1-s}$.

Generic Rules: We add two rules to the set associated to each predicate. These rules are generic and model natural constraints that play an important role in our fact checking system.

The first rule ensures that $p(x,y)$ and $negp(x,y)$ cannot be true at the same time, i.e., a claim should not be assessed as false and true. This is a hard rule, which is always valid.

$$\alpha : \bot \leftarrow p(x,y), negp(x,y) \quad (4)$$

The second rule enforces the *functionality* of a predicate. If a predicate is functional, such as the predicate expressing the capital of a country, then there is only one value that can be in the solution. However, this is not true for all predicates, e.g., a person can be the author of several books. The support of the rule models the functionality of the predicate. We express this constraint stating that a claim cannot have two different object values.

$$w: \quad \bot \leftarrow p(x,y), p(x,z), y \neq z \qquad (5)$$

These generic rules steer the reasoner in the computation of the truthfulness/falseness probability for the input claim.

3.2 Evidence Generation

For a given claim, we execute the following steps to gather the evidence for a fact checking decision.

Generate Evidence Triples from KG: For each rule in Σ, we substitute the head variables with the values of the claim and collect all the triples in the KG that have a valid substitution to its body. More precisely, the head variables in the body of a rule are constrained to the value of the subject and object of the claim. Then, the *evidence triples* are identified by querying the KG with the rewritten body of the rule. For example, given the *spouse* rule above and claim *spouse*(Mike,Laure), the body is rewritten as a query: `child(Mike,c)`, `parent(c,Laure)`, where *c* is a universal variable.

Generate Evidence Triples from Web: Our reasoner models also the uncertainty for the evidence facts. The KG is considered trustworthy, so the weights for the evidence from the KG are set at infinite. However, because of the OWA, we cannot find every true fact in the KG. For claims for which no rule can be executed, we resort to a Web text mining system (Popat et al., 2016). For each rule, we substitute the subject and the object according to the input claim. If a single atom is non-replaceable with KG facts in the body of a rule, then we use the Web module to validate the missing fact. Notice that only grounded facts can be verified with the Web module, such as *child*(Mike,Marie). If the rewritten body contains a fact with a variable, such as *child*(Mike,*c*) above, we discard the claim. If the Web module returns a probability *p* of a fact being correct greater than 0.5, than we add it to our evidence.

As an example, consider the positive rule: *locatedIn(x,y) ← hasCapital(z,x), locatedIn(x,y),*

the claim *locatedIn*(Sacramento, USA), and a KG with fact *hasCapital*(CA, Sacramento). Assuming that the fact for CA located in USA is missing from the KG, we query the Web module for *locatedIn*(CA, USA).

Similarly to the conversion of the rule support into the weight of an LP$^{\text{MLN}}$ program (Section 3.1), we convert the probability *p* of a fact of being true into a weight *w* for the fact when we use it as evidence for the reasoner.

3.3 Inference for Fact Checking

We discuss two inference methods that enable us to expose the rules and the evidence triples involved in a decision for a claim $p(x,y)$.

- **Pure ASP** checks if $p(x,y)$ or $negp(x,y)$ is in the stable model of the rules without including the rule weights. This method only states if the positive or negative triple for the claim can be derived. Since we rely on Horn rules, there is only one stable model for them. If the stable model contains both $p(x,y)$ and $negp(x,y)$, it violates constraint (4), so we conclude neither $p(x,y)$ nor $negp(x,y)$. A similar case happens when the stable model violates the functionality of a predicate.

- LP$^{\text{MLN}}$ **MAP inference with weighted rules** checks if $p(x,y)$ or $negp(x,y)$ is in the most probable stable model of the weighted rules using LPMLN2ASP. This method utilizes the weighted rules and the evidence facts to find a more likely answer at the cost of violating constraints (4) and (5).

Example. We want to check if *Glen Cook* is the author of the book *Cold Copper Tears*. The following weighted rules are mined from the KG[2]:

```
1 0.04: author(A,B) ← runtime(A,C),
    activeYearsStartYear(B,D), C<D.
2 0.04: author(A,B) ← birthYear(B,C),
    runtime(A,D), C>D.
3 0.13: author(A,B) ← author(C,B),
    subsequentWork(A,C).
4 0.02: author(A,B) ← previousWork(A,C),
    literaryGenre(C,D),genre(B,D).
5 0.02: negauthor(A,B) ← writer(C,B),
    format(C,D), format(A,D).
6 0.38: negauthor(A,B) ← runtime(A,C),
    activeYearsStartYear(B,D), C<D.
7 0.31: negauthor(A,B) ← birthYear(B,C),
    runtime(A,D), C>D.
8 0.02: negauthor(A,B) ← writer(C,B),
    previousWork(C,A).
```

---

[2]For readability, we report normalized support (confidence) for rules (evidence triples), instead of weights.

```
 9  0.02: negauthor(A,B) ← writer(C,B),
        previousWork(C,D), subsequentWork(A,D).
10  0.08: negauthor(A,B) ← writer(C,B),
        genre(C,D), genre(A,D).
11  0.02: negauthor(A,B) ← writer(C,B),
        subsequentWork(C,A).
12  0.02: negauthor(A,B) ← previousWork(A,C),
        subsequentWork(D,C), writer(D,B).
13    α: ⊥ ← negauthor(A,B), author(A,B).
14  0.04: ⊥ ← author(A,B), author(A,C), B≠C.
```

Notice that not all rules are semantically correct: rule 1 is not valid (and has low support), while rule 3 is correct in most cases (in fact it has a higher support). Notice also rule 13, which is the hard constraint stating that a fact cannot be true and false at the same time and rule 14 reflecting the low functionality for the *author* predicate. The evidence generator module collects the following triples from the KG (facts with confidence 1) and the Web mining module (all other facts):

```
0.55: literaryGenre('Cold_Copper_Tears','
   Fantasy').
0.52: literaryGenre('Cold_Copper_Tears','
   Mystery_fiction').
1: previousWork('Cold Copper Tears','Bitter Gold Hearts').
0.69: subsequentWork('Cold Copper Tears','Old Tin
   Sorrows').
0.56: activeYearsStartYear('Glen_Cook
   ','1970').
0.59: author('Bitter_Gold_Hearts','Glen_Cook
   ').
1: author('Old Tin Sorrows','Glen Cook').
1: genre('Glen Cook','Fantasy').
1: genre('Glen_Cook','Science_fiction').
1: literaryGenre('Bitter_Gold_Hearts','
   Mystery_fiction').
1: literaryGenre('Bitter Gold Hearts','Fantasy').
1: literaryGenre('Old_Tin_Sorrows','
   Mystery_fiction').
1: literaryGenre('Old_Tin_Sorrows','Fantasy')
   .
1: previousWork('Bitter_Gold_Hearts','
   Sweet_Silver_Blues').
1: previousWork('Old_Tin_Sorrows','
   Cold_Copper_Tears').
1: subsequentWork('Bitter_Gold_Heart','
   Cold_Copper_Tears').
1: subsequentWork('Old_Tin_Sorrows','
   Dread_Brass_Shadows').
1: author('The_Black_Company','Glen_Cook').
1: genre('The_Black_Company','Dark_fantasy').
1: genre('The_Black_Company','Epic_fantasy').
1: genre('The_Black_Company','Fantasy_novel')
   .
```

The LP^{MLN} inference outputs that the input fact is true because of rules 3 and 4 together with the facts in bold in the evidence set. Here, *Old Tin Sorrows* is the *subsequentWork* of *Cold Copper Tears* whose author is *Glen Cook*. These two facts satisfy the body of rule 3 to derive the *author* relation between *Cold Copper Tears* and *Glen Cook*. Similarly, for rule 4, *Fantasy* is the *genre* of *Glen*

Cook, which is also the *literaryGenre* of book *Bitter Gold Hearts*. Further, *Bitter Gold Hearts* is the *previousWork* of *Cold Copper Tears*. This sequence of three facts in the evidence set satisfies the body of rule 4 to derive the *author* relation between the test entities. By using the MAP inference, we can find in the answer set:

```
author(Cold_Copper_Tears,Glen_Cook)
```

4 Experiments

We test our proposal against baseline methods over claims from a real KG. Code and datasets are available online[3].

	spouse	*deathPl.*	*vicePres.*	*almaMater*
Positive	22	25	65	27
Negative	72	33	27	21

Table 2: Number of discovered rules for each predicate.

Datasets. From the latest (online) DBpedia, we selected four predicates *P* = *spouse*, *deathPlace*, *vicePresident*, *almaMater*. In the following, all rules have been mined from 2K positive and 2K negative examples. Statistics for the discovered rules are reported in Table 2.

We create 3 datasets with each containing 100 true and 100 false facts for every predicate, for a total of 2400 claims. True facts are randomly taken from the KG, false ones are created according to the procedure described in the previous section. True facts are then removed from the graph.

Metrics. For each claim in the 4 predicates, we count *correctly* labelled claims (T) and *incorrectly* labelled ones (F). We also count *Undecided* (U) claims, when a method cannot make a decision, e.g., neither *p(x,y)* or *negp(x,y)* are in the stable model. We use precision, defined as (T)/(T+F), recall, defined as (T)/(T+F+U), and their combination in the F-score (harmonic mean).

Methods. We run three baseline methods. The first is the Web text miner CREDE (Popat et al., 2016). Although CREDE was not designed to check arbitrary claims, we show that it also handles KG facts reasonably well. The second is the state of the art link prediction method for KGs KGM (Shi and Weninger, 2016), which uses the graph facts as training data and an ML classifier. The third baseline is the application of the discovered rules, without considering their weights

---

[3]https://github.com/ppapotti/expclaim

(ASP). The first two approaches (CREDE and KGM) cannot provide explanations, while the third (ASP) does not exploit the reasoning. We identified 0.5 as the threshold value for both CREDE and KGM to maximize their F-score.

We consider two variants of our solution. The first is the LP$^{\text{MLN}}$ MAP inference with weighted rules over the KG data only (MAP). The second is MAP with evidence collected from the KG and Web documents (MAP+W). For these methods, we check if the claim is in the stable model.

	almaMat.	*deathPl.*	*spouse*	*vicePres.*
CREDE	.41(.03)	.59(.06)	.44(.07)	.36(.15)
KGM	.73(.08)	.68(.01)	.86(.01)	**.81**(.03)
ASP	.70(.06)	.01(.01)	.31(.08)	.18(.16)
MAP	**.88**(.14)	.75(.15)	**.87**(.11)	.66(.22)
MAP+W	**.88**(.09)	**.83**(.11)	.86(.10)	.68(.18)

Table 3: Average F-score results (SD) for four predicates with all methods over 3 datasets.

Results. Table 3 reports F-score results and standard deviation (SD) for true and false claims averaged over the 3 datasets. For two predicates, MAP+W is the best method in terms of F-score, with an average over all predicates of 0.81, followed by MAP with .79 and KGM with .77. For all predicates, method ASP has very poor performance because of a large number of claims with no rule to be grounded with the KG evidence. Several of these claims are solved with the reasoner in MAP with high precision (1 in most cases) but not perfect recall. Web evidence in MAP+W enables the triggering of more rules, but at the cost of a lower precision because the text miner is not always reliable, as shown in the results for CREDE.

The issue of undecided claims affects the results heavily for predicate *vicePresident* in all methods based on rules. In general, there is no clear correlation between the number of rules and the quality of the results for the rule-based methods. This suggests that the quality of the rules (and of the evidence facts) is more important than their number. Also, more functional predicates, such as *spouse*, are easier to fact check for most methods.

Table 6 reports a detailed analysis for predicate *deathPlace*. The first evidence is that KGM has the best performance for true claims but falls behind MAP methods for false ones. Neither CREDE performs well with false claims. We emphasize that in fact checking false claims are more important.

Results for the rule-based methods show that reasoning is key for our approach. For true claims,

```
FALSE : almaMater(Michael White, UT Austin)
    ← employer(Michael White, UT Austin)
    ← occupation(Michael White, UT Austin)
    ← almaMater(Michael White, Abilene
        Christian Univ.), almaMater(Michael
        White, Yale Divinity School)
```

Table 4: Example of MAP+W output for claim *almaMater*(Michael White, UT Austin).

ASP correctly labels only 1% of the test facts, while the MAP labels 58% of them without mistakes on average. ASP cannot handle facts for which there is a contradiction among the positive and the negative rules, while MAP inference makes the right decision by exploiting the weights of the rules. However, for 42 true claims on average, none of the rules are triggered in MAP. The coverage is increased by adding more evidence with the Web mining module (MAP+W), at the cost of a lower precision but better overall F-score. The benefit of rules and Web evidence is more apparent with false claims. While in this setting CREDE and KGM show poor results, MAP+W reports high precision (94% on average) and an average recall of 83%, with a very significant increase in all metrics compared to MAP. From a manual verification, we explain the better results for false claims with the better quality of the negative rules w.r.t. positive ones for *deathPlace*, i.e., it is easier to find a negative rule than a positive rule for this predicate. This is consistent with previous rule quality assessments (Ortona et al., 2018).

In all the cases for which an answer is produced, rule-based methods explain their decision by showing involved rules and corresponding evidence sets. This makes it relatively easy to identify what is the cause for a conclusion, as for the example reported in Table 4. The given claim is labeled as false because of the three rules that apply with evidence coming both from the KG and the Web.

	spouse	*deathPl.*	*vicePres.*	*almaMat.*
CREDE	6435	7377	7210	7355
KGM	16	15	12	13
ASP	7	8	9	8
MAP	475	822	1880	408
MAP+W	485	1897	3448	409

Table 5: Average execution times (secs) for 200 claims.

Finally, we report on the execution times in Table 5. Methods KGM and ASP are the fastest, with a single claim checked in less than 0.1 seconds. Although we are not counting the time to

	True claims					False claims				
	CREDE	KGM	ASP	MAP	MAP+W	CREDE	KGM	ASP	MAP	MAP+W
Correct(/100)	50(8)	96(1)	1(1)	58(27)	62(31)	23(8)	7(2)	0	62(14)	78(8)
Incorrect(/100)	19(3)	4(1)	0	0	21(18)	55(1)	93(2)	0	11(4)	5(4)
Undecided(/100)	31(9)	0	99(1)	42(27)	17(13)	22(7)	0	100(1)	28(18)	17(9)
Precision	.72	.96	1	1	.75	.29	.07	1	.85	.94
Recall	.69	1	.01	.58	.83	.78	1	0	.72	.83
F-score	.70	.98	.01	.74	.79	.43	.13	.01	.78	.88

Table 6: Average results (SD) for *deathPlace* predicate with all methods over 3 datasets.

gather Web pages, CREDE is the slowest method, with up to 37 seconds on average to check a claim. MAP and MAP+W are in the middle, taking from 2 to 17 seconds to check a claim on average. The time differences depend on the predicate at hand, as checking predicates with less evidence in KG requires more calls to the text mining module.

5 Related Work

There are two main tasks in computational fact checking: (1) monitor and spot claims (Hassan et al., 2017; Jaradat et al., 2018), (2) check claims and explain outcomes. We focus on the second task and on factual facts, specifically. Related approaches try to align the fact to trusted data resources, such as KGs (Shiralkar et al., 2017; Huynh and Papotti, 2018), Web documents (Lehmann et al., 2012), and databases (Cao et al., 2018; Wu et al., 2014). These approaches create features for binary classifiers from the data in the KG. Features exploit the structure of the training examples, in the form of paths (Shi and Weninger, 2016; Ciampaglia et al., 2015) or geometric properties in a multi-dimensional space with embeddings (Bordes et al., 2013; Socher et al., 2013). We distinguish from these works by providing semantically rich rules and evidence facts as explanations for a fact checking outcome.

Markov Logic combines first-order logic and Markov networks (Richardson and Domingos, 2006). In principle, learning in Markov Logic could learn the uncertain rules and inference can be applied to the learned rules. We tested *alchemy* to learn rules for *spouse* relation with 10 positive examples and it was not able to produce results after 2 hours of execution. This illustrates that rule learning in Markov Logic has scalability issues with large KGs. ILP systems for rule discovery, such as ALEPH (Srinivasan, 2001), make assumptions on the input data that do not hold in KGs and RUDIK outperforms this kind of systems (Ortona et al., 2018).

Other proposals have studied the problem of explainable fact checking with rules, but they focus on manually crafted constraints (Gad-Elrab et al., 2019; Leblay, 2017), while our system relies on discovered rules only. Experimental results on the same DBpedia predicates reported in previous work (Gad-Elrab et al., 2019) show that our solution performs better despite being fully automatic. Our proposal also does better than similar attempts that rely on KG only (Pradhan, 2018).

6 Conclusion

We presented a fully automated fact checking framework based on KGs and Web documents as reference information. Given a fact expressed as a triple over entities in the KG, our method validates its veracity with high accuracy and provides an explanation of the decision by exposing facts that support or contradict the given claim according to a set of rules. The system does not rely on a human configuration, as rules are automatically discovered and additional information to complement the KG is mined from the Web.

An interesting direction for extending the framework is to include a module for claim detection and explore the opportunities of an end-to-end system (Thorne et al., 2018). A second direction is to exploit the information from the reasoner to steer the quality management of the KG (Dong et al., 2014), e.g., inspect undecided claims to identify parts of the KG that need data curation. Finally, we aim at integrating natural language generation techniques to produce explanations that are easier to read for the target users (Gatt and Krahmer, 2018).

References

BABAKAR, M. AND MOY, W. 2016. The state of automated factchecking. https://fullfact.org/blog/2016/aug/automated-factchecking/.

BORDES, A., USUNIER, N., GARCIA-DURAN, A., WESTON, J., AND YAKHNENKO, O. 2013. Translating embeddings for modeling multi-relational data. In *NIPS*. 2787–2795.

CAO, T. D., MANOLESCU, I., AND TANNIER, X. 2018. Searching for truth in a database of statistics. In *WebDB*. 4:1–4:6.

CIAMPAGLIA, G. L., SHIRALKAR, P., ROCHA, L. M., BOLLEN, J., MENCZER, F., AND FLAMMINI, A. 2015. Computational fact checking from knowledge networks. *PloS one 10,* 6.

DONG, X., GABRILOVICH, E., HEITZ, G., HORN, W., LAO, N., MURPHY, K., STROHMANN, T., SUN, S., AND ZHANG, W. 2014. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *KDD*. 601–610.

FERRARIS, P., LEE, J., AND LIFSCHITZ, V. 2011. Stable models and circumscription. *Artificial Intelligence 175,* 236–263.

GAD-ELRAB, M. H., STEPANOVA, D., URBANI, J., AND WEIKUM, G. 2019. Exfakt: A framework for explaining facts over knowledge graphs and text. In *WSDM*. 87–95.

GALÁRRAGA, L., TEFLIOUDI, C., HOSE, K., AND SUCHANEK, F. M. 2015. Fast rule mining in ontological knowledge bases with AMIE+. *The VLDB Journal 24,* 6, 707–730.

GATT, A. AND KRAHMER, E. 2018. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research 61,* 65–170.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *ICLP*. 1070–1080.

HASSAN, N., ARSLAN, F., LI, C., AND TREMAYNE, M. 2017. Toward automated fact-checking: Detecting check-worthy factual claims by claimbuster. In *KDD*.

HUYNH, V. AND PAPOTTI, P. 2018. Towards a benchmark for fact checking with knowledge bases. In *Companion of The Web Conference 2018 on The Web Conference (WWW)*.

JARADAT, I., GENCHEVA, P., BARRÓN-CEDEÑO, A., MÀRQUEZ, L., AND NAKOV, P. 2018. ClaimRank: Detecting check-worthy claims in Arabic and English. In *NAACL-HTL*. 26–30.

LEBLAY, J. 2017. A declarative approach to data-driven fact checking. In *AAAI*. 147–153.

LEE, J., LIFSCHITZ, V., AND PALLA, R. 2008. A reductive semantics for counting and choice in answer set programming. In *AAAI*. 472–479.

LEE, J., TALSANIA, S., AND WANG, Y. 2017. Computing LPMLN using ASP and MLN solvers. *Theory and Practice of Logic Programming*.

LEE, J. AND WANG, Y. 2016. Weighted rules under the stable model semantics. In *KR*. 145–154.

LEHMANN, J., GERBER, D., MORSEY, M., AND NGOMO, A. N. 2012. Defacto - deep fact validation. In *ISWC*. 312–327.

ORTONA, S., MEDURI, V. V., AND PAPOTTI, P. 2018. Robust discovery of positive and negative rules in knowledge bases. In *ICDE*. 1168–1179.

POPAT, K., MUKHERJEE, S., STRÖTGEN, J., AND WEIKUM, G. 2016. Credibility assessment of textual claims on the web. In *CIKM*.

POPAT, K., MUKHERJEE, S., YATES, A., AND WEIKUM, G. 2018. Declare: Debunking fake news and false claims using evidence-aware deep learning. In *EMNLP*. 22–32.

PRADHAN, A. 2018. Explainable fact checking by combining automated rule discovery with probabilistic answer set programming. M.S. thesis. ASU.

RICHARDSON, M. AND DOMINGOS, P. 2006. Markov logic networks. *Machine learning 62,* 1-2, 107–136.

SHI, B. AND WENINGER, T. 2016. Discriminative predicate path mining for fact checking in knowledge graphs. *Knowledge-Based Systems 104,* 123–133.

SHIRALKAR, P., FLAMMINI, A., MENCZER, F., AND CIAMPAGLIA, G. L. 2017. Finding streams in knowledge graphs to support fact checking. In *ICDM*. 859–864.

SOCHER, R., CHEN, D., MANNING, C. D., AND NG, A. 2013. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*. 926–934.

SRINIVASAN, A. 2001. The Aleph manual.

SUCHANEK, F. M., KASNECI, G., AND WEIKUM, G. 2007. Yago: a core of semantic knowledge. In *WWW*. ACM, 697–706.

THORNE, J., VLACHOS, A., CHRISTODOULOPOULOS, C., AND MITTAL, A. 2018. FEVER: a large-scale dataset for fact extraction and verification. In *NAACL-HLT*.

WU, Y., AGARWAL, P. K., LI, C., YANG, J., AND YU, C. 2014. Toward computational fact-checking. *Proceedings of the VLDB Endowment 7,* 7, 589–600.