Buckle: Evaluating Fact Checking Algorithms Built on Knowledge Bases

Viet-Phi Huynh
Eurecom
viet-phi.huynh@eurecom.fr

Paolo Papotti
Eurecom
papotti@eurecom.fr

ABSTRACT

Fact checking is the task of determining if a given claim holds. Several algorithms have been developed to check facts with reference information in the form of knowledge bases. We demonstrate BUCKLE, an open-source benchmark for comparing and evaluating fact checking algorithms in a level playing field across a range of scenarios. The demo is centered around three main lessons. To start, we show how, by changing the properties of the training and test facts, it is possible to influence significantly the performance of the algorithms. We then show the role of the reference data. Finally, we discuss the performance for algorithms designed on different principles and assumptions, as well as approaches that address the *link prediction task* in knowledge bases.

PVLDB Reference Format:
Huynh, Papotti. BUCKLE: Evaluating Fact Checking Algorithms Built on Knowledge Bases. *PVLDB*, 12(12): 1798 - 1801, 2019.
DOI: https://doi.org/10.14778/3352063.3352069

1. INTRODUCTION

While fact checking has historically been an activity for journalists, the increase of incorrect claims over the Web has motivated the study of computational methods to identify misleading claims. In fact, manual assessment of facts (as done by journalists in websites such as politifact.com) cannot scale with the proliferation of sources spreading false information [9]. Different efforts tackle different types of facts and domains. We focus on algorithms that test textual claims against trustful Knowledge Bases (KBs), such as "Leo Tolstoy won the Nobel Prize". We assume that entities and predicates in "worth checking" facts have been identified [5], and study the step estimating the *veracity* of a fact (expressed as structured data) w.r.t. trusted reference data.

A core issue for fact checking with KBs is that the reference information is incomplete (Open World Assumption), i.e., a fact not in the KB can either be false or just missing [3]. Given a KB K and a fact f, our fact checking task is to state if f is a valid missing fact in K, a problem that can be seen as a special case of link prediction in graphs [2].

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352069

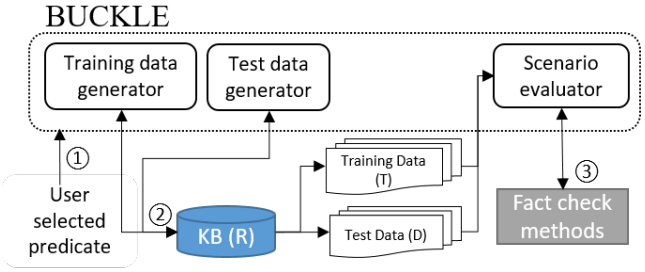

Figure 1: The benchmark architecture.

Automatic fact checking algorithms come from different intuitions. Some of them rely on paths and sub-graphs in the KB: they assume that training examples are available and learn models to label new facts to be tested. Other approaches assume that a set of constraints over the KB have been discovered and can be exploited to validate or invalidate a given fact. Others rely on embeddings to model a candidate predicate between two entities as a translation in the corresponding low dimensional vector space.

Due to the richness and diversity of algorithms, it is important to conduct fair experimental evaluations to assess the potential of each proposal. Thorough evaluation of fact checking algorithms requires systematic control over the training and test data, and the quality of the KBs used as reference information. To support rigorous empirical evaluations, a fact generation system must be able to generate multiple scenarios, composed of true and false claims, with low user effort and clear evaluation results.

BUCKLE is the first scenario generator conceived to support empirical evaluations of fact checking algorithms as per the above requirements. We show the major components of the system in Figure 1. A user starts interacting with BUCKLE by selecting a predicate from a KB (e.g., *married* for DBpedia) and the complexity of the scenario. The system creates a scenario with real data, according to the given configuration, in terms of training, test, and reference data [6]. The scenario is executed with multiple algorithms and their results are exposed for comparison.

The demonstration will convey three primary insights about benchmarking fact checking algorithms.

(1) We introduce data properties that characterize the difficulty of a scenario. We will show how enforcing such properties on training and test data radically affects the results of the fact checking step.

(2) Then, we will discuss how the characteristics of the reference KBs may significantly influence the quality of the results by an algorithm.
(3) Finally, we will demonstrate seven different algorithms in action on scenarios generated by using BUCKLE, to reveal new insights on their performance.

The control over the data features distinguishes the experiments enabled by BUCKLE from previous efforts in evaluating fact checking algorithms. The attendees will see how the availability of a benchmark levels the field and raises the bar for evaluation standards in this task. BUCKLE is open-source (`https://github.com/huynhvp/BUCKLE-Fact_checking`), so that it can be further extended by the community.

The paper is organized as follows. Section 2 introduces the background for the benchmark. Section 3 provides an overview of the system and the data properties at its core. Finally, Section 4 discusses the organization of the demo.

2. FACT CHECKING ALGORITHMS

Background. A *fact* is defined as a triple that has the form of ("subject" s, "predicate" p, "object" o). Natural language processing techniques are used to convert a textual claim into a structured format. Facts can be classified into categories, such as numerical, quote, and object property. We focus on object properties, which are facts stating a relationship between the subject and the object in a sentence, e.g., Sacramento is the capital of California.

A *Knowledge Base* (KB) is a direct graph where nodes correspond to entities (subject or object in a fact) and edges model binary predicates among entities. We focus on algorithms taking as input a KB and a fact that is not part of it. Such algorithms assess if the fact belongs to the missing part of the KB (therefore is "true") or no (is "false"). Most entities in a KB have a predicate defining their type (e.g., general as "thing", or specific as "person" or "company").

Fact checking algorithms. Algorithms assume that either training examples (labelled facts) or reference information (the KB itself) are available to build the internal models for checking claims. A common assumption is that the KB is trustable, and training examples are derived from it. However, algorithms are very different in the way they model the problem, as we describe next.

Structure Based Algorithms. Given a fact (s, p, o), this group of algorithms makes a decision for it by exploiting the topological structures identified in the KB by the p triples. The KB triples are used to learn the alternative paths (different from p) between their subjects and objects. Properties of the paths are then modeled as features in a classifier that decides if predicate p holds for the given s and o.

Knowledge Linker (**KL**) builds a weighted adjacency matrix with edge weights computed as the in-degree of each node in the KB [2]. The model ignores predicate labels and evaluates the validity of a fact based on the proximity between its subject and object. Every path connecting a given subject and object is then mapped to a score computed on the frequency of the nodes in the KB.

Discriminate Predicate Path Mining (**KG-Miner**) exploits frequent predicate paths between pair of entities in the KB [8]. Given a test fact (s, p, o) and examples P that satisfy p in the KB, it collects the predicate paths for every node pair in P having subject with the same type of s (denoted $\varphi(s)$) and object with type of o ($\varphi(o)$). From each $u \in \varphi(s)$ and corresponding $v \in \varphi(o)$, predicate paths that alternatively represent predicate p are extracted by traversing the graph from u to v. The path information gain is computed based on the number of occurrences. It then selects the most discriminative paths as features and train a logistic regression model to classify positive and negative examples. The model is then used to compute the likelihood of a test fact.

Path Ranking Algorithm (**PRA**) extracts features from a training triple with random walks starting at the source and at the corresponding target node to retrieve the paths between them [4]. Top k paths for each instance are collected in a feature matrix. A value in the matrix for a training instance (s, p, o) is the probability of arriving at the target node o by a random walk starting at source node s by following one of its top k paths. The feature matrix is then used with a classifier to validate the input fact.

Sub-graph Feature Extraction (**SFE**) extends PRA by extracting features from KB sub-graphs [4]. The *sub-graph of depth m* for each node n is the result of m breadth-first search steps from it. A sequence of predicates connecting source and target nodes is obtained by intersecting their sub-graphs. SFE uses such sequences to identify *binary features* that disregard the frequency (KG-Miner) or the probability (PRA) of feature paths. Features are then used in a classifier trained on positive and negative examples.

Embedding Based Algorithms. Embeddings encode entities and predicates in the KB into a low-dimensional vector space while preserving certain information of the graph and minimizing a margin-based ranking loss. A predicate in the graph is interpreted as a translation from subject entity to object entity in such space. To check a fact (s, p, o), these methods check the relevance of the embedding representations $\mathbf{s}$ of s and $\mathbf{o}$ of o w.r.t. embedding representation $\mathbf{p}$ of predicate p though a specific score function.

Translating Embeddings (**TransE**) represents a predicate p from triple (s, p, o) as a translation from subject s to object o on the same low-dimensional embedding space, that is $\mathbf{s} + \mathbf{p} \approx \mathbf{o}$ if (s, p, o) is true [1]. Its score function is defined as: $f(s, p, o) = \|\mathbf{s} + \mathbf{p} - \mathbf{o}\|$, where $\|.\|$ can be either L1 or L2 norm. As TransE uses the same embedding space for both entities and predicates, a many-to-one predicate can lead to identical embedding representations for different subject entities in it. To address this issue, **TransH** enables an entity to have different embedding representations w.r.t. the different predicates it participates. For each predicate p, it introduces a specific hyperplane w_p (normal vector) and defines embedding vector $\mathbf{p}$ on this hyperplane.

Rule based algorithms. A fact can be validated or invalidated by using a set of rules defined over the KB (**Rules**). For example, a "positive" rule can state that for every city that is the capital of a country, that city is also located in the same country, or a "negative" rule can state that if someone has a father relation with another person, it is unlikely that they are married. Such rules Σ can be manually defined or discovered from a KB [3, 7]. Given a set of rules Σ, a fact is validated if it matches a positive rule and it is invalidated if it matches at least one negative rule or has no match.

3. THE BENCHMARK

Fact checking scenario. Any benchmark has a set of standard application scenarios that can be tested against different systems sharing similar functionalities. This implies that

every scenario is correctly interpreted by the systems. Unlike benchmarks for other data-centric applications, such as query processing for DBMS, there is no definition or common practice for a fact checking scenario. In our benchmark, a fact checking scenario is defined for a predicate p with a triple (R, T, D), where R is the reference data (the KB), T is the training data (true and false p facts w.r.t. R), and D is the test data (true and false p facts, missing from R).

BUCKLE consists of three components, as depicted in Figure 1. A user interacts with BUCKLE by selecting any predicate in a KB (1). For the input predicate, two system components combine the real data to generate T and D according to the configuration (2). A third component combines and executes the scenario over the target algorithms (3).

Reference data. Our system can work with any RDF knowledge base, but for the sake of simplicity we focus the description on DBPedia, a KB with triples extracted from Wikipedia. From these triples, we construct a graph by assigning each unique entity to a graph node, and converting the triple into a directed edge with label "predicate" from the entity "subject" to entity "object". We obtain a directed graph with ≈4M nodes, ≈27M edges, and 671 predicates. We treat the input KB as trusted (i.e., assumed mostly correct) but incomplete (open world assumption).

The reference data R plays a crucial role in all algorithms. Intuitively, the true fact that "Sacramento is the capital of California" is easier to automatically check compared to the true fact that "Kinshasa is the capital of Zaire". In fact, even if both facts are missing in the KB, the two entities in *capital(Sacramento,California)* are more popular in most KBs than the African city and country. This can be captured by analyzing R. We enforce this data quality in a scenario by distinguishing facts based on their *context* information. In particular, we characterize the entities based on their structural properties. We define the *popularity* $\mathcal{G}(x)$ of an entity x as the number of incoming and outcoming edges for its node in the KB graph. For a given fact $p(s, o)$, we then compute the popularity of its pair of entities (s, o) by computing the average of their popularity scores.

Table 1: Examples of claims (false ones in italic)

	Dataset	Example Facts
Functional	Easy 1	Arizona, capital, Phoenix
		Arizona, capital, Oregon
	Easy 2	Massachusetts, capital, Boston
		Massachusetts, capital, Worcester
	Medium 1	Massachusetts, capital, Boston
		Massachusetts, capital, Worcester
		France, capital, Paris
		Japan, capital, Osaka
Non functional	Medium 2	Never Go Back, author, Lee Child
		Personal (novel), author, Lee Child
		Greater Journey, author, Michael Lewis
	Hard 1	J. Kittinger, award, War Prisoner Medal
		J. Kittinger, award, Bronze Star Medal
		H. F. Davison, award, Military Cross
		J. L. Morgan, award, Military Cross
		Lothar Linke, award, 2009 RTHK

Training and test data. The training and test data generators take as input distinct parameters and produce distinct sets, but are based on common methods. Each training T and test D dataset includes both true and false facts for a specific predicate, as reported in the examples shown in Table 1. We distinguish three independent properties for the data, and build scenarios that have different levels of difficulty based on how such properties are set and combined.

We use predicate *capital* to illustrate the dataset properties. Capital is a one-to-one (functional) predicate, which contains facts for cities and states: *capital(city,state)*.

The first property is the **transparency**. Consider a scenario with 50 US capitals as true facts either in training or test data. From these 50 correct triples, we can generate 200 incorrect triples by random matching each capital to 4 other states, as shown for the dataset "Easy 1". In this scenario, true/false facts lead to clear internal representation with any model because the entities in the negative examples have little in common in R. However, this changes with *more challenging false facts*. Unlike "Easy 1", in which false instances are created by random matching capital cities to states, example "Easy 2" contains as false facts triples from another predicate among cities and states (*largestCities*). For each state in the true capital-state triples, we can include its largest cities as false triples in the datasets. Since the large cities and the capital cities share some properties in R, they are "less transparent" and harder to distinguish by the algorithms. For a predicate p, this property is controlled by setting the percentage of false facts from type-compatible pairs connected by *any* predicate different from p.

The second property is the **homogeneity**. We started with a scenario with *capital* triples for 50 US states, as this is a setting studied in several papers in the literature. But, in such a well scoped scenario, all entities are semantically close to each other. That is, to check whether a US city in D is capital of a US state, the algorithms rely on the information from other US capitals and cities in T. In reality, data is oftentimes heterogeneous, e.g., covering both US and European cities. To break homogeneity, we can use capitals of world countries, so that the facts model a more general concept of *capital* (example "Medium 1"). For a predicate p, this property is controlled by clustering the pairs of entities with the embedding methods presented in the previous section. We then determine the difficulty of the scenario based on the size of the selected cluster (the smaller, the easier).

The third property characterizes the **functionality** of the predicate. A one-to-one predicate is easier to model than a one-to-many predicate, such as persons in the *author* relationship with at least one book, but each book has at most one author (example "Medium 2"). The hardest case are many-to-many predicates, such as *award*, i.e., persons who got a prize or a recognition. Notice also how in example "Hard 1" the examples are not homogeneous, as people from very different backgrounds got awards in different domains.

Our scenario generator has default configurations that mix the properties above to obtain scenarios of different difficulties. For example, a functional predicate chosen by the user needs higher percentage of non homogeneous and non transparent examples to reach the Medium difficulty compared to a non functional predicate. Users can change these percentages from the GUI, as described in the next section.

4. DEMONSTRATION SCENARIO

The audience will interact with the system with two main interfaces. The *scenario generation* is reported in Figure 2, while the *algorithm execution* is reported in Figure 3.

Scenario generation. In the interface in Figure 2, the users will be asked to select the KB and the predicate to

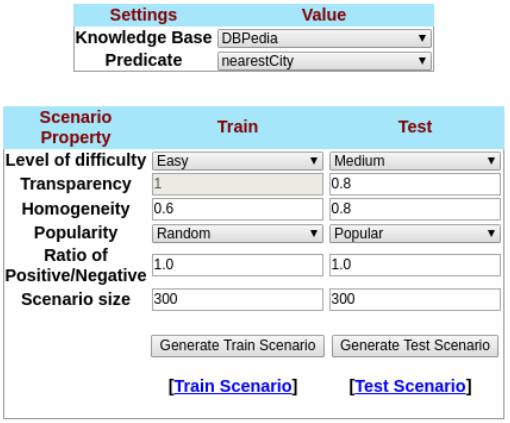

Figure 2: The benchmark input screen.

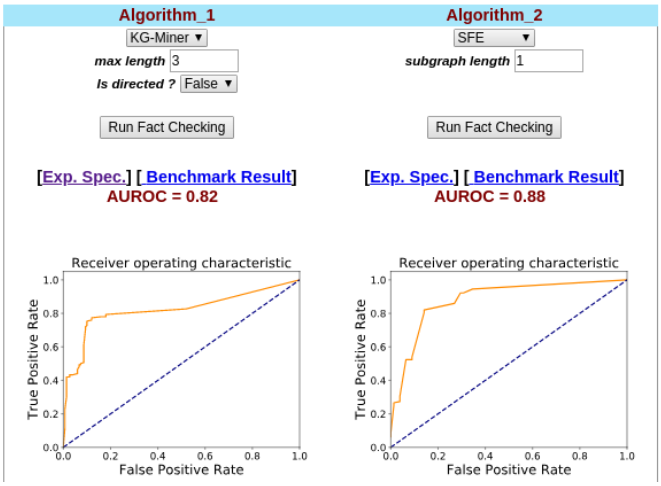

Figure 3: Result screen for two algorithms executed on the same scenario.

test. The different settings for a scenario include the data properties discussed in the previous section.

Algorithm execution. Once a scenario has been generated, we will ask the users to select the algorithms to be executed (top part of Figure 3). The systems can be tuned with the algorithm-specific parameters that dynamically appear by interacting with the menus. For example, by selecting algorithm **KL**, the choice between *metric* and *ultra-metric* closure appears as an option. For quality evaluation of the results, we use the Area Under the Receiver Operating Characteristic curve (AUROC).

Demo outline. We will start by asking the users to generate scenarios with the default levels of difficulty, such as Easy and Hard. The users will see the generated training and test datasets obtained by combining the real data from the KB at hand. Once a scenario has been generated, we will ask the users to select one algorithm (with its default parameters) and run it. The lesson here is about the role of data. Users will see how, by changing our data properties only from an Easy to Hard scenario, the same algorithm goes from good accuracy results to random guessing.

In a second interaction, we will ask the users to freely change the parameters in both interfaces and observe how results for the same algorithm change. The goal is to make evident to the audience that a single data property has a bigger impact on the qualitative results than any algorithmic choice. More specifically, while any method performs well in a simple, functional predicate scenario, qualitative differences in the results of all algorithms become evident with datasets involving non functional predicates, more ambiguous training and test datasets, and less popular entities. Users will also be able to replicate experimental settings from papers in the literature.

Finally, we will invite the users to execute two algorithms on the same dataset, as shown in Figure 3. The goal is to demonstrate how our benchmark can lead to useful insights about advantages and limits of the different methods.

Users will be able to see differences across methods from the same family and from different ones. For example, algorithm **KL** is more robust to nonfunctional predicates than other path-based methods, as it does not rely on the predicate semantics expressed by the labels. On the other hand, **KL** has issues in the classification of scenarios where false instances are created from semantically close but different predicates. For example, Worcester is not *capital* of Massachusetts, but it is one of its *largestCities*. Due to the lack of semantics, **KL** treats both Worcester and Boston as capital of Massachusetts because it finds good proximity between the two cities and the state in the graph.

As an example of differences across families, users will be able to see how path based methods, such as **KG-Miner**, **PRA**, and **SFE**, can outperform **TransE** and **TransH**, but also show a more significant drop in quality for medium and hard scenarios. In these cases, increasing the default length of predicate paths leads to more discriminative and informative features, thereby improving the performance but also increasing the execution times. As another example, by changing the *Popularity* parameter in the generator, users can see how embedding and rule based methods (**Rules**) are more sensible than path based algorithms to the quality of the reference data.

5. REFERENCES

[1] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

[2] G. L. Ciampaglia, P. Shiralkar, L. M. Rocha, J. Bollen, F. Menczer, and A. Flammini. Computational fact checking from knowledge networks. *PloS one*, 10(6):e0128193, 2015.

[3] L. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.*, 24(6):707–730, 2015.

[4] M. Gardner and T. M. Mitchell. Efficient and expressive knowledge base completion using subgraph feature extraction. In *EMNLP*, pages 1488–1498, 2015.

[5] N. Hassan, F. Arslan, C. Li, and M. Tremayne. Toward automated fact-checking: Detecting check-worthy factual claims by claimbuster. In *KDD*, 2017.

[6] V. Huynh and P. Papotti. Towards a benchmark for fact checking with knowledge bases. In *Companion of the TheWebConf (WWW)*, pages 1595–1598, 2018.

[7] S. Ortona, V. Meduri, and P. Papotti. Robust discovery of positive and negative rules in KBs. In *ICDE*, 2018.

[8] B. Shi and T. Weninger. Discriminative predicate path mining for fact checking in knowledge graphs. *Knowledge-Based Systems*, 104:123–133, 2016.

[9] X. Wang, C. Yu, S. Baumgartner, and F. Korn. Relevant document discovery for fact-checking articles. In *Companion of the TheWebConf (WWW)*, 2018.