DeFacto - Deep Fact Validation

Jens Lehmann, Daniel Gerber, Mohamed Morsey, and Axel-Cyrille Ngonga Ngomo

Universität Leipzig, Institut für Informatik, AKSW,
Postfach 100920, D-04009 Leipzig, Germany,
{lehmann|dgerber|morsey|ngonga}@informatik.uni-leipzig.de
http://aksw.org

Abstract. One of the main tasks when creating and maintaining knowledge bases is to validate facts and provide sources for them in order to ensure correctness and traceability of the provided knowledge. So far, this task is often addressed by human curators in a three-step process: issuing appropriate keyword queries for the statement to check using standard search engines, retrieving potentially relevant documents and screening those documents for relevant content. The drawbacks of this process are manifold. Most importantly, it is very time-consuming as the experts have to carry out several search processes and must often read several documents. In this article, we present DeFacto (Deep Fact Validation) – an algorithm for validating facts by finding trustworthy sources for it on the Web. DeFacto aims to provide an effective way of validating facts by supplying the user with relevant excerpts of webpages as well as useful additional information including a score for the confidence DeFacto has in the correctness of the input fact.

1 Introduction

The past decades, in particular due to the rise of the World Wide Web, have marked a change from an industrial society to an information and knowledge society. Creating and managing knowledge successfully has been a key to success in various communities worldwide. Therefore, the quality of knowledge is of high importance. One aspect of knowledge quality is provenance. In particular, the sources for facts should be well documented, since this provides several benefits such as a better detection of errors, decisions based on the trustworthiness of sources etc. While provenance is an important aspect of data quality [8], to date only few knowledge bases actually provide provenance information. For instance, less than 3% of the more than 607.7 million RDF documents indexed by Sindice[1] contain metadata such such as creator, created, source, modified, contributor, or provenance.[2] This lack of provenance information makes the validation of the facts in such knowledge bases utterly tedious. In addition, it hinders the adoption of such data in business applications as the data is not trusted [8]. The main

[1] http://www.sindice.com
[2] Data retrieved on June 6, 2012.

contribution of this paper is the provision of a fact validation approach and tool which can make use of one of the largest sources of information: the Web.

More specifically, our system DeFacto (Deep Fact Validation) implements algorithms for validating statements, specifically RDF triples, by finding confirming sources for it on the web. It takes a statement as input (e.g., that shown in listing 1) and then tries to find evidence for the validity of that statement by searching for textual information in the web. In contrast to typical search engines, it does not just search for textual occurrences of parts of the statement, but tries to find webpages which contain the actual statement phrased in natural language. It presents the user with a confidence score for the input statement as well as a set of excerpts of relevant webpages, which allows the user to manually judge the presented evidence.

DeFacto has two major use cases: (1) Given an existing true statement, it can be used to find provenance information for it. For instance, the WikiData project[3] aims to create a collection of facts, in which sources should be provided for each fact. DeFacto could be used to achieve this task. (2) It can check whether a statement is likely to be true, provide the user with a confidence score in whether the statement is true and evidence for the score assigned to the statement. Our main contributions are thus as follows: (1) An approach that allows checking whether a webpage confirms a fact, i.e., an RDF triple, (2) an adaptation of existing approaches for determining indicators for trustworthiness of a webpage, (3) an automated approach to enhancing knowledge bases with RDF provenance data at triple level as well as (4) a running prototype of DeFacto, the first system able to provide useful confidence values for an input RDF triple given the Web as background text corpus.

The rest of this paper is structured as follows: Section 2 describes our general approach and the system infrastructure. The next section describes how we extended the BOA framework to enable it to detect facts contained in textual descriptions on webpages. In Section 4, we describe how we include the trustworthiness of webpages into the DeFacto analysis. Section 5 combines the previous chapters and describes the mathematical features we use to compute the confidence for a particular input fact. We use those features to train different classifiers in Section 6 and describe our evaluation results. Section 7 summarizes related work. Finally, we conclude in Section 8 and give pointers to future work.

2 Approach

Input and Output: The DeFacto system consists of the components depicted in Figure 1. The system takes an RDF triple as input and returns a confidence value for this triple as well as possible evidence for the fact. The evidence consists of a set of webpages, textual excerpts from those pages and meta-information on the pages. The text excerpts and the associated meta information allow the user to quickly get an overview over possible credible sources for the input statement: Instead of having to use search engines, browsing several webpages and looking for

[3] `http://meta.wikimedia.org/wiki/Wikidata`

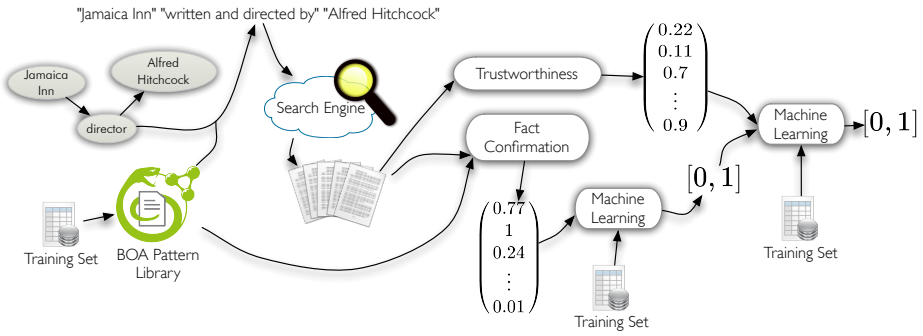

Fig. 1. Overview of Deep Fact Validation.

relevant pieces of information, the user can more efficiently review the presented information. Moreover, the system uses techniques which are adapted specifically for fact validation instead of only having to rely on generic information retrieval techniques of search engines.

Retrieving Webpages: The first task of the DeFacto system is to retrieve webpages which are relevant for the given task. The retrieval is carried out by issuing several queries to a regular search engine. These queries are computed by verbalizing the RDF triple using natural-language patterns extracted by the BOA framework[4] [5, 4]. Section 3.2 describes how the search engine queries are constructed. As a next step, the highest ranked webpages for each query are retrieved. Those webpages are candidates for being sources for the input fact. Both the search engine queries as well as the retrieval of webpages are executed in parallel to keep the response time for users within a reasonable limit. Note that usually this does not put a high load on particular web servers as webpages are usually derived from several domains.

Evaluating Webpages: Once all webpages have been retrieved, they undergo several further processing steps. First, plain text is extracted from each webpage by removing most HTML markup. We can then apply our fact confirmation approach on this text, which is described in detail in Section 3. In essence, the algorithm decides whether the web page contains a natural language formulation of the input fact. This step distinguishes DeFacto from information retrieval methods. If no webpage confirms a fact according to DeFacto, then the system falls back on light-weight NLP techniques and computes whether the webpage does at least provide useful evidence. In addition to fact confirmation, the system computes different indicators for the trustworthiness of a webpage (see Section 4). These indicators are of central importance because a single trustworthy webpage confirming a fact may be a more useful source than several webpages with low

[4] `http://boa.aksw.org`

trustworthiness. The fact confirmation and the trustworthiness indicators of the most relevant webpages are presented to the user.

Confidence Measurement: In addition to finding and displaying useful sources, DeFacto also outputs a general confidence value for the input fact. This confidence value ranges between 0% and 100% and serves as an indicator for the user: Higher values indicate that the found sources appear to confirm the fact and can be trusted. Low values mean that not much evidence for the fact could be found on the Web and that the websites that do confirm the fact (if such exist) only display low trustworthiness. The confidence measurement is based on Machine Learning techniques and explained in detail in Sections 5 and 6. Naturally, DeFacto is a (semi-)automatic approach: We do assume that users will not blindly trust the system, but additionally analyze the provided evidence.

Using the LOD Cloud as Background Knowledge: As described above, DeFacto relies primarily on natural language from several webpages as input. However, in some cases, confirming facts for an input statement can be found in openly available knowledge bases. Due to the fast growth of the LOD cloud, we expect this source to become increasingly important in the future. In order to use this additional evidence, DeFacto provides a component which searches for similar statements in the LOD cloud. To achieve this, the system first finds similar resources to the subject and object of the input triple, which is currently done via the `http://sameas.org` service. In a second step, it retrieves all triples which use the detected similar subject and object resources by dereferencing the corresponding Linked Data URIs. Finally, the labels of subject, predicate and object of all triples are retrieved. Those are then compared via string similarity techniques to the input triple. Currently, the average trigram similarity of subject, predicate and object of the triple is used. In this article, we focus on re-using textual evidence and plan to carry out a more detailed evaluation of the LOD as background knowledge in future work.

```
1 dbpedia-res:Jamaica_Inn_%28film%29 dbpedia-owl:director
2       dbpedia-res:Alfred_Hitchcock .
```

Listing 1. Input data for Defacto.

A prototype implementing the above steps is available at `http://defacto.aksw.org`. It shows relevant webpages, text excerpts and five different rankings per page. In the case, many possible sources for the statement were found, which is why DeFacto displays a high confidence. The generated provenance output can also be saved directly as RDF. For representing the provenance output, we use the W3C provenance group[5] vocabularies. The source code of both, the DeFacto algorithms and user interface, are openly available[6].

It should be noted that we decided not to check for negative evidence of facts in DeFacto, since a) we considered this to be too error-prone and b) negative

[5] `http://www.w3.org/2011/prov/`
[6] `https://github.com/AKSW/DeFacto`

statements are much less frequent on the web. It is also noteworthy that DeFacto is a self training system on two levels: For each fact, the user can confirm after reviewing the possible sources whether he believes it is true. This is then added to the training set and helps to improve the performance of DeFacto. The same can be done for text excerpts of web pages: Users can confirm or reject whether a given text excerpt actually does confirm a fact. Both machine learning parts are explained in Sections 3 and 6.

3 BOA

The idea behind BOA is two-fold: first, it aims to be a framework that allows extracting structured data from the Human Web by using Linked Data as background knowledge. In addition, it provides a library of natural-language patterns that allows to bridge the gap between structured and unstructured data. The input for the BOA framework consists of a set of knowledge bases, a text corpus (mostly extracted from the Web) and (optionally) a Wikipedia dump[7]. When provided by a Wikipedia dump, the framework begins by generating surface forms for all entities in the source knowledge base. The surface forms used by BOA are generated by using an extension of the method proposed in [12]. For each predicate p found in the input knowledge sources, BOA carries out a sentence-level statistical analysis of the co-occurrence of pairs of labels of resources that are linked via p. BOA then uses a supervised machine-learning approach to compute a score and rank patterns for each combination of corpus and knowledge bases. These patterns allow generating a natural language representation of the RDF triple that is to be checked.

3.1 Training BOA for DeFacto

In order to provide a high quality fact confirmation component, we trained BOA specifically for this task. We began by selecting the top 60 most frequently used object properties from the DBpedia [13, 10] ontology using the DBpedia Live endpoint[8]. This query retrieves 7.750.362 triples and covers 78% of the 9.993.333 triples in DBpedia with *owl:ObjectProperty*s from the DBpedia namespace.[9] Currently, we focus on object properties. Adequate support of datatype properties requires an extension of the presented methods, which is planned in future work. For each of those properties, we selected the top 10 BOA patterns (if available) sorted according to the number of triples this pattern has been learned from. This resulted in a list of 488 patterns which were evaluated by all four authors. During this process, each pattern was labeled by two persons independently. We judged a pattern as positive if it was not generic (e.g., "?D? 's " ?R?") or specific enough (e.g., "?D? in the Italian region ?R?") and could be

[7] `http://dumps.wikimedia.org/`

[8] `http://live.dbpedia.org/sparql`

[9] Properties like wikiPageExternalLink, wikiPageRedirects, wikiPageDisambiguates and thumbnail have been excluded.

used to express the relation in natural text. The first group achieved a moderate Cohen's-Kappa value of 0.477 and the second group scored a good value of 0.626. Every conflict was resolved by having the annotators agree on a single annotation. The resulting annotations were use for a 10-fold cross-validation training of BOA's neural network. We achieved the maximum F-score of 0.732 with an error threshold of 0.01 and a hidden layer size of 51 neurons.

3.2 Automatic Generation of Search Queries

The found BOA patterns are used for issuing queries to the search engine (see Figure 1). Each search query contains the quoted label of the subject of the input triple, a quoted and cleaned BOA pattern (we remove punctuation characters which are not indexed be the search engine) and the quoted label of the object of the input triple. We use a fixed number of the best scored BOA patterns whose score was beyond a score threshold and retrieve the first n websites from a web search engine. For our example from Listing 1, such a query is "Jamaican Inn" AND "written and directed by" AND "Alfred Hitchcock". Then, we crawl each website and try to extract *possible proofs* for the input triple in parallel. A possible proof for an input triple in our sense is an excerpt of a webpage, which may confirm it. In the sequel, we just write proof instead of possible proof.

3.3 BOA and NLP Techniques for Fact Confirmation

To find proofs for a given input triple we make use of the surface forms introduced in [12]. We select all surface forms for the subject and object of the input triple and search for all occurrences of each combination of those labels in a website w. If we find an occurrence with a token distance $d(l(s), l(o))$ (where $l(x)$ is the label of x) smaller then a given threshold we call this occurrence a proof for the input triple. To remove noise from the found proofs we apply normalization by regular expression filters which for example remove characters between brackets and non alpha-numeric characters. Note that this normalization facilitates the grouping of proofs by their occurrence. After extracting all proofs $p_i \in \mathcal{P}(w)$ of a website w we score each proof using a linear regression classifier. We trained a classifier with the following input features for scoring a proof:

BOA Pattern: This is a Boolean features which is 1 if a BOA pattern is contained in the normalized proof phrase.

BOA Score: If BOA patterns are found in the normalized proof phrase, then the score of the highest score across the set of found patterns is written in this feature. Else, this feature is set to 0.

Token Distance: This is the distance $d(l(s), l(o))$ between the two entity labels which found the proof. We limit this distance to a maximum of 20 tokens.

Wordnet Expansion: We expand both the tokens of the normalized proof phrase as well as all of the tokens of the BOA pattern with synsets from Wordnet. Subsequently we apply the Jaccard-Similarity on the generated expansions. This is basically a fuzzy match between the BOA pattern and the proof phrase.

Table 1. Performance measures for several classifiers on the fact confirmation task (AUC = area under the ROC curve, RMSE = root mean squared error).

	P	R	F_1	AUC	RMSE
Logistic Regression	0.769	0.769	0.769	0.811	0.4653
Naïve Bayes	0.655	0.624	0.564	0.763	0.5665
SVM	0.824	0.822	0.822	0.823	0.4223
RBFNetwork	0.735	0.717	0.718	0.718	0.485

Total Occurrence: This feature contains the total number of occurrences of each normalized proof phrase over the set of all normalized proof phrases.

Page Title: We apply the maximum of the trigram similarity measure between the page title and the subject and object labels. This feature is useful, because the title indicates the topic of the entire web page. When a title matches, then higher token distances may still indicate a high probability that a fact is confirmed.

End of Sentence: A boolean value if the potential proof contains a ".", "!" or a "?". When subject and object are in different sentences, their relation is more likely to be weaker.

Phrase the words between the subject and object, which are encoded as binary values, i.e. a feature is created for each word and its value is set to 1 if the word occurs and 0 otherwise.

Property the property as a word vector.

To train our classifiers, we randomly sampled 527 proofs and annotated them manually. Those proofs were extracted with DeFacto from applying it on the training set described in Section 6.1. The results are shown in Table 1. We ran popular classifiers, which are able to work with numeric data and create confidence values. The ability to generate confidence values for proofs is useful as feedback for users and it also serves as input for the core classifiers described in Section 6. Based on the obtained results, we selected support vector machines as classifier. We also performed preliminary work on fine-tuning the parameters of the above algorithms, which, however, did not lead to significantly different results. The reported measurements were, therefore, done with default values of the mentioned algorithms in the Weka machine learning toolkit[10] version 3.6.6.

4 Trustworthiness Analysis of Webpages

To determine the trustworthiness of a website we first need to determine its similarity to the input triple. This is determined by how many topics belonging to the query are contained in a search result retrieved by the web search. We extended the approach introduced in [14] by querying Wikipedia with the subject and object label of the triple in question separately to find the topic terms for

[10] `http://www.cs.waikato.ac.nz/ml/weka/`

the triple. A frequency analysis is applied on all returned documents and all terms above a certain threshold that are not contained in a self-compiled stop word list are considered to be topic terms for a triple. Let s and o be the URIs for the subject and object of the triple in question and t be a potential topic term extracted from a Wikipedia page. In addition, let $X = (s, p, o)$.We compare the values of the following two formulas:

$$p(t|X) = \frac{|topic(t, d(X))|}{|d(X)|},$$
$$p(t|intitle(d(X), s \vee o)) = \frac{|topic(t, intitle(d(X), s) \cup intitle(d(X), o))|}{|intitle(d(X), s) \cup intitle(d(X), o)|}.$$

where $d(X)$ is the set all web documents retrieved for X (see Section 3.2), $intitle(d(X), x)$ the set of web documents which have the label of the URI x in their page title. $topic(t, d(X))$ is the set of documents which contain t in the page body. We consider t to be a topic term for the input triple if $p(t|t(d(X), s) \vee t(d(X), o)) > p(t|X)$. Let $\mathcal{T}_X = \{t_1, t_2, \ldots, t_n\}$ be the set of all topic terms extracted for a input triple. Defacto then calculates the trustworthiness of a webpage as follows:

Topic Majority in the Web represents the number of webpages that have similar topics to the webpage in question. Let $\mathcal{P}$ be the set of topic terms appearing on the current webpage. The Topic Majority in the Web for a webpage w is then calculated as:

$$tm_{web}(w) = \left| \bigcup_{i=1}^{n} topic(t_i, d(X)) \right| - 1$$

where t_1 is the most occurring topic term in the webpage w. Note that we subtract 1 to prevent counting w.

Topic Majority in Search Results calculates the similarity of a given webpage for all webpages found for a given triple. Let w_k be the webpage to be evaluated, $v(w_k)$ be the feature vector of webpage w_k where $v(w_k)_i$ is 1 if t_i is a topic term of webpage w_k and 0 otherwise, $\|v\|$ be the norm of v and θ a similarity threshold. We calculate the Topic Majority for the search results as follows:

$$tm_{search}(w) = \left| \left\{ w_i | w_i \in d(X), \frac{v(w_k) \times v(w_i)}{\|v(w_k)\| \, \|v(w_i)\|} > \theta \right\} \right|$$

Topic Coverage measures the ratio between all topic terms for X and all topic terms occurring in w:

$$tc(w) = \frac{|\mathcal{T}_X \cap \mathcal{P}|}{|\mathcal{T}_X|}$$

Pagerank: The Pagerank[11] of a webpage is a measure for the relative importance of a webpage compared to all others, i.e. higher pageranks means that a webpage is more popular. There is a positive correlation between popularity of a webpage and its trustworthiness as those pages are more likely to be reviewed by more people or may have gone under stricter quality assurance before their publication.While a high pagerank alone is certainly not a sufficient indicator for trustworthiness, we use it in combination with the above criteria in DeFacto.

5 Features for Deep Fact Validation

In order to obtain an estimate of the confidence that there is sufficient evidence to consider the input triple to be true, we decided to train a supervised machine learning algorithm. Similar to the above presented classifier for fact confirmation, this classifier also requires computing a set of relevant features for the given task. In the following, we describe those features and why we selected them.

First, we extend the score of single proofs to a score of web pages as follows: When interpreting the score of a proof as the probability that a proof actually confirms the input fact, then we can compute the probability that at least one of the proofs confirms the fact. This leads to the following stochastic formula[12], which allows us to obtain an overall score for proofs scw on a webpage w:

$$scw(w) = 1 - \prod_{pr \in prw(w)} (1 - fc(pr))$$

In this formula, fc (fact confirmation) is the classifier trained in Section 3.3, which takes a proof pr as input and returns a value between 0 and 1. prw is a function taking a webpage as input and returning all possible proofs contained in it.

Combination of Trustworthiness and Textual Evidence In general, the trustworthiness of a webpage and the textual evidence we find in it, are orthogonal features. Naturally, webpages with high trustworthiness and a high score for its proofs should increase our confidence in the input fact. Therefore, it makes sense to combine trustworthiness and textual evidence as features for the underlying machine learning algorithm. We do this by multiplying both criteria and then using their sum and maximum as two different features:

$$F_{fsum}(t) = \sum_{w \in s(t)} (f(w) \cdot scw(w)) \qquad F_{fmax}(t) = \max_{w \in s(t)} (f(w) \cdot scw(w))$$

In this formula f can be instantiated by all four trustworthiness measures: topic majority on the the web (tm_{web}), topic majority in search results (tm_{search}),

[11] `http://en.wikipedia.org/wiki/Pagerank`

[12] To be exact, it is the complementary even to the case that none of the proofs does actually confirm a fact.

topic coverage (tc) and pagerank (pr). s is a function taking a triple t as argument, executing the search queries explained in Section 3.2 and returning a set of webpages. Using the formula, we obtain 8 different features for our classifier, which combine textual evidence and different trustworthiness measures.

Other Features In addition to the above described combinations of trustworthiness and fact confirmation, we also defined other features:

1. The total number of proofs found.
2. The total number of proofs found above a relevance threshold of 0.5. In some cases, a high number of proofs with low scores is generated, so the number of high scoring proofs may be a relevant feature for learning algorithms.
3. The total evidence score: This is the probability that at least one of the proofs is correct, which is defined analogously to scw above:

$$1 - \prod_{pr \in prt(t)} (1 - sc(pr))$$

4. The total evidence score above a relevance threshold of 0.5. This is an adaption of the above formula, which considers only proofs with a confidence higher than 0.5.
5. Total hit count: Search engines usually estimate the number of search results for an input query. The total hit count is the sum of the estimated number of search results for each query send by DeFacto for a given input triple.
6. A domain and range verification: If the subject of the input triple is not an instance of the domain of the property of the input triple, this violates the underlying schema, which should result in a lower confidence in the correctness of the triple. This feature is 0 if both domain and range are violated, 0.5 if exactly one of them is violated and 1 if there is no domain or range violation.

6 Evaluation

Our main objective in the evaluation was to find out whether DeFacto can effectively distinguish between true and false input facts. In the following, we describe how we trained DeFacto using DBpedia, which experiments we used and then discuss the results of those experiments.

6.1 Training DeFacto

As mentioned in Section 3, we focus our tests on the top 60 most frequently used properties in DBpedia. The system can easily be extended to cover more properties by extending the training set of BOA to those properties. Note that DeFacto itself is also not limited to DBpedia, i.e. while all of its components are trained on DBpedia, the algorithms can be applied to arbitrary URIs. A performance evaluation on other knowledge bases is subject to future work, but

it should be noted that most parts of DeFacto – except the LOD background feature described in Section 2 and the schema checking feature in Section 3.3 – work only with the retrieved labels of URIs and, therefore, do not depend on DBpedia.

For training a supervised machine learning approach, positive and negative examples are required. Those were generated as follows:

Positive Examples: In general, we use facts contained in DBpedia as positive examples. For each of the properties we consider (see Section 3), we generate positive examples by randomly selecting triples containing the property. Technically, this is done by counting the frequency of the property and sending a corresponding SPARQL query with random offset to the DBpedia Live endpoint. We obtain 150 statements this way and verified them. For each statement, we manually evaluated whether it was indeed a true fact. It turned out that some of the obtained triples were incorrectly modeled, e.g. obviously violated domain and range restrictions, or could not be confirmed by an intensive search on the web within ten minutes. Overall, 122 out of 150 checked triples were facts, which we subsequently used as positive examples.

Negative Examples: The generation of negative examples is more involved than the generation of positive examples. In order to effectively train DeFacto, we considered it essential that many of the negative examples are similar to true statements. In particular, most statements should be meaningful subject-predicate-object phrases. For this reason, we derive the negative examples from positive examples by modifying them, but following domain and range restrictions. Assume the input triple (s, p, o) in a knowledge base $\mathcal{K}$ is given and let dom and ran be functions returning the domain and range of a property.[13] We used the following methods to generate negative examples:

1. A triple (s', p, o) is generated where s' is an instance of $dom(p)$, the triple (s', p, o) is not contained in $\mathcal{K}$ and s' is randomly selected from all resources which satisfy the previous requirements.
2. A triple (s, p, o') is generated analogously by taking $ran(p)$ into account.
3. A triple (s', p, o') is generated analogously by taking both $dom(p)$ and $ran(p)$ into account.
4. A triple (s, p', o) is generated in which p' is randomly selected from our previously defined list of 60 properties and (s, p', o) is not contained in $\mathcal{K}$.
5. A triple (s', p', o') is generated where s' and o' are randomly selected resources, p' is a randomly selected property from our defined list of 60 properties and (s', p', o') is not contained in $\mathcal{K}$.

Note that all parts of the example generation procedure can also take implicit knowledge into account. Since we used SPARQL as query language for implementing the procedure, this is straightforward by using SPARQL 1.1 entail-

[13] Technically, we used the most specific class, which was explicitly stated to be domain and range of a property, respectively.

ment[14]. In case of DBpedia Live we did not do this for performance reasons and because it would not alter the results in that specific case.

Obviously, it is possible that our procedure of generating negative examples may also generate true statements, which just happen not to be contained in DBpedia. Similar to the analysis of the positive examples, we also checked whether the negative examples are indeed false statements. Overall, we obtained an automatically created and manually cleaned training set, which we made publicly available[15].

6.2 Experimental Setup

In a first step, we computed all feature vectors, described in Section 5 for the training set. DeFacto heavily relies on web requests, which are not deterministic, i.e. the same search engine query does not always return the same result. To achieve deterministic behavior and to increase performance and reduce load on the server, all web requests are cached. The DeFacto runtime for an input triple was on average slightly below 5 seconds per input triple[16] when using caches.

We stored the features in the arff file format and employed the Weka machine learning toolkit[17] for training different classifiers. In particular, we are interested in classifiers, which can handle numeric values and output confidence values. Naturally, confidence values for facts such as, e.g. 95%, are more useful for end users than just a binary response on whether DeFacto considers the input triple to be true, since they allow a more fine-grained assessment. Again, we selected popular machine learning algorithms satisfying those requirements.

We performed 10 fold cross validations for our experiments. In each experiment, we used our created positive examples, but varied the negative example sets described above to see how changes influence the overall behavior of DeFacto. Finally, we performed a run using a mixture of all 5 negative examples sets by randomly selecting 20% of their instances.

Table 2. Classification results for trainings sets *domain* and *range*.

	Domain					Range				
	P	R	F_1	AUC	RSME	P	R	F_1	AUC	RMSE
Logistic Regression	0.821	0.787	0.781	0.789	0.3965	0.846	0.799	0.792	0.762	0.3948
Naïve Bayes	0.63	0.582	0.54	0.599	0.6427	0.775	0.635	0.582	0.68	0.6039
SVM	0.818	0.758	0.746	0.758	0.4917	0.824	0.758	0.745	0.758	0.4917
RBFNetwork	0.675	0.586	0.526	0.625	0.4548	0.711	0.627	0.586	0.654	0.4622

[14] `http://www.w3.org/TR/sparql11-entailment/`

[15] `http://aksw.org/projects/DeFacto`

[16] The performance is roughly equal on server machines and notebooks, since the web requests dominate.

[17] `http://www.cs.waikato.ac.nz/ml/weka/`

Table 3. Classification results for trainings sets *domain-range* and *property*.

	Domain-Range					Property				
	P	R	F_1	AUC	RSME	P	R	F_1	AUC	RMSE
Logistic Regression	0.833	0.775	0.764	0.783	0.4045	0.775	0.766	0.765	0.768	0.4317
Naïve Bayes	0.782	0.652	0.606	0.688	0.5902	0.454	0.48	0.395	0.707	0.7181
SVM	0.822	0.754	0.74	0.754	0.4959	0.793	0.787	0.786	0.785	0.4111
RBFNetwork	0.763	0.656	0.617	0.688	0.4589	0.62	0.615	0.61	0.686	0.4722

Table 4. Classification results for trainings sets *random* and *random-20%mix*.

	Combined Negative Examples					Random 20% Mix				
	P	R	F_1	AUC	RMSE	P	R	F_1	AUC	RMSE
Logistic Regression	0.711	0.701	0.697	0.742	0.4593	0.894	0.893	0.893	0.925	0.2967
Naïve Bayes	0.641	0.557	0.48	0.599	0.6525	0.667	0.635	0.617	0.757	0.5709
SVM	0.638	0.619	0.605	0.724	0.4657	0.891	0.889	0.889	0.889	0.3326
RBFNetwork	0.602	0.578	0.552	0.652	0.4878	0.812	0.799	0.797	0.852	0.3823

6.3 Results and Discussion

The results of our experiments are shown in Tables 2-4. Two algorithms – logistic regression and support vector machines – show promising results. Given the challenging tasks, F-measures up to 70% for the combined negative example set appear to be very positive indicators that DeFacto can be used to effectively distinguish between true and false statements, which was our primary evaluation objective. In general, DeFacto also appears to be quite stable against the various negative example sets: The algorithms with overall positive results also seem less affected by the different variations. As expected, the easiest task is to distinguish random statements and true statements, whereas all other replacements are similarly difficult.

When observing single runs of DeFacto manually, it turned out that our method of generating positive examples is particularly challenging for DeFacto: For many of the facts in DBpedia only few sources exist in the Web. While it is widely acknowledged that the amount of unstructured textual information in the Web by far superseeds the available structured data, we found out that a significant amount of statements in DBpedia is difficult to track back to reliable external sources on the Web even with an exhaustive manual search. There are many reasons for this, for instance many facts are particular relevant for a specific country, such as "Person x studied at University y.", where x is a son of a local politician and y is a country with only limited internet access compared to first world countries. For this reason, only in some cases, BOA patterns could be found: In 29 of the 527 proofs of positive examples, BOA patterns could directly be found. This number increased to 195 out of 527 when employing the WordNet expansion described in Section 3.3. In general, DeFacto performs better when the subject and object of the input triple are popular on the web, i.e. there are

several webpages describing them. In this aspect, we believe our training set is indeed challenging upon manual observation.

7 Related Work

While we are not aware of existing work in which sources for RDF statements were detected automatically from the Web, there are three main areas related to DeFacto research: The representation of provenance information in the Web of Data as well as work on trustworthiness and relation extraction. The problem of data provenance is a crucial issue in the Web of Data. While data extracted by the means of tools such as Hazy[18] and KnowItAll[19] can be easily mapped to primary provenance information, most knowledge sources were extracted by non-textual source and are more difficult to link with provenance information. In the work described in [9], Olaf Hartig and Jun Zhao developed a framework for provenance tracking. This framework provides the vocabulary required for representing and accessing provenance information on the web. It keeps track of who created a web entity, e.g. a webpage, when it was last modified etc. Recently, a W3C working group has been formed and released a set of specifications on sharing and representing provenance information[20]. Dividino et al. [3] introduced an approach for managing several provenance dimensions, e.g. source, and timestamp. In their approach, they described an extension to the RDF called RDF$^+$, which can efficiently work with provenance data. They provided a method to extend SPARQL query processing in a manner such that a specific SPARQL query can request meta knowledge without modifying the query itself. Theoharis et al. [18] argued how the implicit provenance data contained in a SPARQL query results can be used to acquire annotations for several dimensions of data quality. They detailed the abstract provenance models and how they are used in relational data, and how they can be used in semantic data as well. Their model requires the existence of provenance data in the underlying semantic data source. DeFacto uses the W3C provenance group standard for representing provenance information. Yet, unlike previous work, it directly tries to find provenance information by searching for confirming facts in trustworthy webpages.

The second related research area is trustworthiness. Nakamura et al. [14] developed an efficient prototype for enhancing the search results provided by a search engine based on trustworthiness analysis for those results. They conducted a survey in order to determine the frequency at which the users accesses search engines and how much they trust the content and ranking of search results. They defined several criteria for trustworthiness calculation of search results returned by the search engine, such as topic majority. We adapted their approach for DeFacto and included it as one of the features for our machine learning techniques. [16, 17] present an approach for computing the trustworthiness of web pages. To achieve this goal, the authors rely on a model based on hubs and authorities.

[18] `http://hazy.cs.wisc.edu/hazy/`
[19] `http://www.cs.washington.edu/research/knowitall/`
[20] `http://www.w3.org/2011/prov/wiki/`

This model allows to compute the trustworthiness of facts and websites by generating a k-partite network of pages and facts and propagating trustworthiness information across it. The approach returns a score for the trustworthiness of each fact. An older yet similar approach is that presented in [20]. Here, the idea is to create a 3-partite network of webpages, facts and objects and apply a propagation algorithm to compute weights for facts as well as webpages. The use of trustworthiness and uncertainty information on RDF data has been the subject of recent research (see e.g., [7, 11]). Our approach differs from these approaches as it does not aim to evaluate the trustworthiness of facts expressed in natural language. In addition, it can deal with the broad spectrum of relations found on the Data Web.

Our approach is also related to relation extraction. Most tools that address this task rely on pattern-based approaches. Some early work on pattern extraction relied on supervised machine learning [6]. Yet, such approaches demanded large amounts of training data, making them difficult to adapt to new relations. The subsequent generation of approaches to RE aimed at bootstrapping patterns based on a small number of input patterns and instances. For example, [2] presents the Dual Iterative Pattern Relation Expansion (DIPRE) and applies it to the detection of relations between authors and titles of books. This approach relies on a small set of seed patterns to maximize the precision of the patterns for a given relation while minimizing their error rate of the same patterns. Snowball [1] extends DIPRE by a new approach to the generation of seed tuples. Newer approaches aim to either collect redundancy information (see e.g., [19] in an unsupervised manner or to use linguistic analysis [15] to harvest generic patterns for relations.

8 Conclusion and Future Work

In this paper, we presented DeFacto, an approach for checking the validity of RDF triples using the Web as corpus. We showed that our approach achieves an F_1 measure around 0.8 on DBpedia. Our approach can be extended in manifold ways. First, BOA is able to detect natural-language representations of predicates in several languages. Thus, we could have the user choose the languages he understands and provide facts in several languages, therewith also increasing the portion of the Web that we search through. Furthermore, we could extend our approach to support data type properties. In addition to extending our approach by these two means, we will also focus on searching for negative evidence for facts, therewith allowing users to have an unbiased view of the data on the Web through DeFacto. On a grander scale, we aim to provide even lay users of knowledge bases with the means to check the quality of their data by using natural language input. This would support the transition from the Document Web to the Semantic Web by providing a further means to connect data and documents.

References

1. Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *In ACM DL*, pages 85–94, 2000.
2. Sergey Brin. Extracting patterns and relations from the world wide web. In *WebDB*, pages 172–183, London, UK, 1999. Springer-Verlag.
3. Renata Dividino, Sergej Sizov, Steffen Staab, and Bernhard Schueler. Querying for provenance, trust, uncertainty and other meta knowledge in rdf. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3), 2011.
4. Daniel Gerber and Axel-Cyrille Ngonga Ngomo. Extracting multilingual natural-language patterns for rdf predicates. In *Proceedings of EKAW*, 2012.
5. Daniel Gerber and Axel-Cyrille Ngonga Ngomo. Bootstrapping the linked data web. In *1st Workshop on Web Scale Knowledge Extraction @ ISWC 2011*, 2011.
6. R. Grishman and R. Yangarber. Nyu: Description of the Proteus/Pet system as used for MUC-7 ST. In *MUC-7*. Morgan Kaufmann, 1998.
7. Olaf Hartig. Trustworthiness of data on the web. In *Proceedings of the STI Berlin CSW PhD Workshop*, 2008.
8. Olaf Hartig. Provenance information in the web of data. In *Proceedings of LDOW*, 2009.
9. Olaf Hartig and Jun Zhao. Publishing and consuming provenance metadata on the web of linked data. In *IPAW*, pages 78–90, 2010.
10. Jens Lehmann, Chris Bizer, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009.
11. Timm Meiser, Maximilian Dylla, and Martin Theobald. Interactive reasoning in uncertain RDF knowledge bases. In Bettina Berendt, Arjen de Vries, Weinfei Fan, and Craig Macdonald, editors, *CIKM'11*, pages 2557–2560, 2011.
12. Pablo N. Mendes, Max Jakob, Andres Garcia-Silva, and Christian Bizer. DBpedia Spotlight: Shedding Light on the Web of Documents. In *I-SEMANTICS*, ACM International Conference Proceeding Series, pages 1–8. ACM, 2011.
13. Mohamed Morsey, Jens Lehmann, Sören Auer, Claus Stadler, and Sebastian Hellmann. Dbpedia and the live extraction of structured data from wikipedia. *Program: electronic library and information systems*, 46:27, 2012.
14. Satoshi Nakamura, Shinji Konishi, Adam Jatowt, Hiroaki Ohshima, Hiroyuki Kondo, Taro Tezuka, Satoshi Oyama, and Katsumi Tanaka. In *ECDL*, volume 4675, pages 38–49, 2007.
15. Dat P. T. Nguyen, Yutaka Matsuo, and Mitsuru Ishizuka. Relation extraction from wikipedia using subtree mining. In *AAAI*, pages 1414–1420, 2007.
16. Jeff Pasternack and Dan Roth. Generalized fact-finding. In *WWW '11*, pages 99–100, 2011.
17. Jeff Pasternack and Dan Roth. Making better informed trust decisions with generalized fact-finding. In *IJCAI*, pages 2324–2329, 2011.
18. Yannis Theoharis, Irini Fundulaki, Grigoris Karvounarakis, and Vassilis Christophides. On provenance of queries on semantic web data. *IEEE Internet Computing*, 15:31–39, 2011.
19. Yulan Yan, Naoaki Okazaki, Yutaka Matsuo, Zhenglu Yang, and Mitsuru Ishizuka. Unsupervised relation extraction by mining wikipedia texts using information from the web. In *ACL*, ACL '09, pages 1021–1029, 2009.
20. Xiaoxin Yin, Jiawei Han, and Philip S. Yu. Truth discovery with multiple conflicting information providers on the web. In *KDD '07*, pages 1048–1052, 2007.