

# Truthfulness Verification System

Tathagata Dasgupta, ABM Musa  
Email: {tdasgu2, amusa2}@uic.edu  
URL: <http://code.google.com/p/tverifier>

## 1 Introduction

Web has become the most prevalent source of information now a days. However, many information on the web is untruthful. Also because of the widespread reach of web, sometimes web is used to propagate untruthful facts for social and political reasons. Hence with the increasingly use of web as information source, verification of information truthfulness became an important facet to consider.

The popular search engines extract information from web based on keywords and metadata without considering the truthfulness of the facts. Also, there have been not much research in this area. To our best knowledge, the most recent work on this area is T-verifier [5], which uses results from search engines to verify truthfulness of statements. T-verifier performs very well on the test-dataset. However, T-verifier has some problems with the overall approach to verify the truthfulness of statements and there is room for improvement. In this work, we will extend the T-verifier system so that it's weaknesses can be resolved to get a more robust truthfulness verification system.

## 2 Current System

Current system for truthfulness verification is called T-verifier [5], which uses two phase methods for truthfulness verification of statements. Each of these two phases rely heavily on search results returned by popular search engines. T-verifier takes the doubtful statements (*DS*) as input from the user along with the doubtful unit (DU). Phrase after removing DU from *DS* is called topic unit (TU).

At the first phase, T-verifier generates alternative statements by supplying *TU* to search engine and collecting the relevant alternate *DU*-s. However, from basic web search may result in lot of alternate *DU*-s that are not semantically or logically relevant to the original *DS*. Hence, T-verifier uses combination of seven features to rank alternate *DU*-s. These features primarily exploit the facts that relevant alternative units frequently co-occur, people often mention both misconception and truthful conception together, data-type matching, and sense-closeness. T-verifier chooses top 5 alternative statements based on top 5 alternate *DU*-s obtained in this phase.

At the second phase, top 5 alternative statements from phase 1 is supplied to the search engine again. Then the returned searched result is ranked by multiple rankers such as Alternative Unit Ranker, Hits ranker, Text-feature Ranker, Domain Authority Ranker etc. Then all those ranks are

merged to form an overall ranking among the alternate statements and top statement in this final merged ranking is considered as truthful statement.

### 3 Problems with Current System

Although results from the tested dataset achieves good performance (90% accuracy), failing of T-verifier for some statements shows that it has some inherent problems and there is room for improvement.

First, T-verifier assumes that truthful statements will be more propagated in the web compared to the untruthful statement. However, this may be not true in general because of intended and planned propaganda for establishing some untruthful statements. This kind of propagandas are even becoming more common now a days due to widespread reach of Internet. T-verifier also showed that "*Hillary Clinton is the President of United States*" has more hits than "*Hillary Clinton is the Secretary of State*". Although T-verifier was able to find the correct statement in this case using multiple ranks together, in general the untruthful statement can be prevalent in the web compared to truthful statements.

Second, T-verifier do not use the reputation of the information source. T-verifier uses only search results returned by the search engine irrespective of the origin and believability of the information origin. Hence there is room for improvement here to give more weight to the information obtained from trustworthy sources such as Wikipedia.

### 4 Proposed System

We have tried two approaches for using the Wikipedia. In both we utilize the content words to identify articles and there upon treat them separately. In the first approach we treat the articles as bag of words and do not consider the sentence structure and in the next approach we retain the logical structure of each sentence and try to infer from word overlaps.

#### 4.1 Bag of words

In the bag of words method, we are exploiting the fact that for true sentence we will have high content word match between pages obtained from topic unit and alternate units. Overall steps of this method are described below:

- For all possible k-grams ( $k = 1, 2, 3, \dots, \text{topic\_unit\_length}$ ) of topic unit we find relevant wikipedia page using wikipedia search API. Although it may seems that taking all possible k-grams will introduce noise in the retrieved data, this is not the case because there will be no pages for irrelevant k-gram consisting of irrelevant combination of words. Now for each possible k-gram we may get a Wikipedia page directly or we may go need to disambiguate if that k-gram refers to multiple possible pages in Wikipedia. If we get a single page without any ambiguation then we take that page. However, if we need to disambiguate then we look into all possible wikipedia pages and disambiguate them using highest intersection of words of topic unit and description of the wikipedia page for the disambiguation link. Here we take all

words except the k-gram for which we are disambiguating as a set of words for intersection. So assume that at the end of this step we get  $n$  wikipedia pages either directly or through disambiguation.

- Now for each of the five alternative units generated by the T-verifier we also find a wikipedia page. Here also we may get a Wikipedia page directly or we may need to disambiguate if there are multiple possible pages. If we need disambiguation then we disambiguate using same approach described previously i.e. maximal intersection of words between the description of disambiguation link and words of topic unit.
- We have five Wikipedia pages for each of the alternative units now and  $n$  pages from topic unit. We generate bag of words i.e. content words for all the five pages for alternate units and  $n$  pages for topic unit.
- Now for each of the page corresponding to alternate units we count intersection of words for each  $n$  pages for topic unit and take sum of intersection count for an alternate unit page and  $n$  topic unit page. Hence at this step we have a count value associated with each of the alternate unit and we rank all the alternate units according to this value to produce truth confidence or rank for all five alternate units.

Pseudo-code for Bag of words ranking is given below. Listing 1 produces the ranking for all five alternative units following the procedure described above. It uses Listing 2 to generate k-grams and find relevant Wikipedia pages and Listing 3 to disambiguate among alternative pages using intersection of words between topic unit and description of disambiguation link for alternative page

---

**Listing 1** Bag of Words (Ranking Algorithm 1)

---

**Description:** Given the alternate units generated by the current T-verifier system, rank the units in descending order of truthfulness using Wikipedia

**Input:** alter\_units.txt generated by T-verifier. Each line in this file is a tuple containing the sentence id, the alternate units generated and the topic unit.

**Output:** Ranked alter\_units

```
for each sentence  $s_i$  do
   $tu_{s_i}$  = topic unit string
   $s_i.au\_list$  = alternate units generated by T-verifier
   $s_i.tu\_list$  = find_wikiarticles.tu( $tu_{s_i}$ )
  for each alternate unit  $s_i.au_j$  in  $s_i.au\_list$  do
     $w_{s_i.au_j}$  = find_wikiarticles.au( $s_i.au_j$ )
    if  $w_{s_i.au_j} == \phi$  then
      skip to processing next  $w_{s_i.au_j}$ 
    end if
    if  $w_{s_i.au_j}$  is diambiguation page then
       $w_{s_i.au_j}$  = disambiguate( $w_{s_i.au_j}$ )
       $bow_{s_i.au_j}$  = generate_bag_of_words( $s_i.au_j$ )
    end if
  end for
  for each  $w_{s_i.tu_j}$  in  $s_i.tu\_list$  do
     $bow_{s_i.tu_j}$  = generate_bag_of_words( $s_i.tu_j$ )
  end for
  for  $s_i.au_j$  in  $s_i.au\_list$  do
    for each  $w_{s_i.tu_k}$  in  $s_i.tu\_list$  do
       $common\_words_{tu_k}^{au_j} = bow_{s_i.au_j} \cap bow_{s_i.tu_k}$ 
       $remaining\_words_{s_i.tu_k} = tu_{s_i}$  - words forming title of  $w_{s_i.tu_j}$  - stop words
      for each word  $w$  in  $remaining\_words_{s_i.tu_k}$  do
         $s_i.au_j.score$  += count of  $w$ 
      end for
    end for
  end for
  print  $s_i.au_j$  reverse sorted by  $s_i.au_j.score$ 
end for
```

---

---

**Listing 2** Find wiki articles (called from Bag of Words)

---

**Description:** Given a string of words determine phrases that may be titles of wikipedia articles

**Input:** a string of words  $s$

$l = \text{length}(s)$

$result = \phi$

**for**  $i$  in 1 to  $l$  **do**

**for**  $j$  in  $i$  to  $l$  **do**

$sub\_string_{ij} += s_i + blank$

**end for**

**if**  $sub\_string_{ij}$  consists only stopwords **then**

        discard and continue to next iteration

**else**

        search wikipedia with  $sub\_string_{ij}$

$result_{sub\_string_{ij}} = (\text{displaytitle}, \text{url}, \text{categories}, \text{redirects})$  for  $sub\_string_{ij}$

**end if**

**end for**

**return** result

---

---

**Listing 3** Disambiguate articles (called from Bag of Words)

---

**Description:** Given a disambiguation page returns the wikipedia article that is most relevant to sentence being processed  
**Input:** disambiguation page  $p, tu_{s_i}$   
extract all outlinks urls from the body of the page  
form a dictionary  $d$  using disambiguation element ( $de$ ) and the disambiguation description ( $dd$ ) appearing adjacent to the  $de$   
**for** every  $de$  in  $d$  **do**  
     $de.ddscore = \text{count content word overlap between } dd \text{ and } (tu_{s_i} - \text{words in } de)$   
**end for**  
**return** top ranked  $de$

---

**Example** Detail working of Bag of Words (Algorithm 1) for following sentence is given below:

*Les Paul invented the electric guitar*

For this sentence, the alternative units are les paul, gibson les, llyod, leo, adolph rickenbacker. At first, we generate all possible k-grams from the topic unit. The generated k-grams are following:

['invented', 'Invented', 'invented the', 'Invented The', 'invented the electric', 'Invented The Electric', 'invented the electric guitar', 'Invented The Electric Guitar', 'the', 'The', 'the electric', 'The Electric', 'the electric guitar', 'The Electric Guitar', 'electric', 'Electric', 'electric guitar', 'Electric Guitar', 'guitar', 'Guitar']

Now for each k-gram we find relevant wikipedia page. We will not get any wikipedia page for irrelevant k-gram combination. Moreover, We may get a page directly or we may need to disambiguate. Following are the all relevant pages. The False in the second column page there is no ambiguation and True means we need to disambiguate.

```
http://en.wikipedia.org/wiki/Invention, False
http://en.wikipedia.org/wiki/The_Electric, True
http://en.wikipedia.org/wiki/Electric_guitar, False
http://en.wikipedia.org/wiki/Electricity, False
http://en.wikipedia.org/wiki/Electric_guitar, False
http://en.wikipedia.org/wiki/Guitar, False
```

For the above pages, we need to disambiguate for one page only and that is [http://en.wikipedia.org/wiki/The\\_Electric](http://en.wikipedia.org/wiki/The_Electric). We described earlier that we look into the count of overlap between topic unit words and disambiguation link description words. For this disambiguation there is no overlap and so we do not need to process this page further. Hence the final set of  $n$  pages are following:

```
http://en.wikipedia.org/wiki/Invention
http://en.wikipedia.org/wiki/Electric_guitar
http://en.wikipedia.org/wiki/Electricity
http://en.wikipedia.org/wiki/Electric_guitar
http://en.wikipedia.org/wiki/Guitar
```

In the next step, we find relevant Wikipedia page for each of the alternate units. There are no relevant Wikipedia page for gibson les and llyod. Hence we do not process these alternative units further. For the remaining three alternative units, there is no ambiguation for les paul and adolph rickenbacker. The corresponding pages are following:

[http://en.wikipedia.org/wiki/Les\\_Paul](http://en.wikipedia.org/wiki/Les_Paul)  
[http://en.wikipedia.org/wiki/Adolph\\_Rickenbacker](http://en.wikipedia.org/wiki/Adolph_Rickenbacker)

However we need to disambiguate for the page corresponding to Leo:

<http://en.wikipedia.org/wiki/Leo>

After disambiguation for Leo using word overlap between description and topic unit words, we get following page:

[http://en.wikipedia.org/w/index.php?action=raw&title=Leo\\_\(constellation\)](http://en.wikipedia.org/w/index.php?action=raw&title=Leo_(constellation))

Now, we need to count word overlap for each of the pages for les paul, adolph rickenbacker, leo with all pages for the topic unit and take sum of the overlap to produce the ranking of alternate units

## 4.2 Sentence Splitting

The main idea behind this ranking algorithm for alternative units is that Wikipedia should contain content words from doubt statement in the same sentence because of the similarity in the context. Details description of the algorithm is given below.

---

**Listing 4** Sentence Structure (Ranking Algorithm 2)

---

```
Search google with topic unit
take first Wikipedia page from google search result
Extract all sentences from the Wikipedia page
for each alternate unit do
  for each sentence from Wikipedia page do
    if alternate unit is found in the sentence then
      count [alternate unit]= no of intersection of content words from topic unit and the sentence
    end if
  end for
end for
rank the alternate units using descending order of count[alternate unit]
```

---

**Examples** Detail working of Sentence Structure (Algorithm 2) for following sentence is given below:

*Les Paul invented the electric guitar*

For this sentence the alternative units are les paul, gibson les, llyod, leo, adolph rickenbacker. The top wikipedia page from google search is [http://en.wikipedia.org/wiki/Electric\\_guitar](http://en.wikipedia.org/wiki/Electric_guitar). So we download this page. Convert it to text from html and split into sentences. Then we match each alternate unit with all the sentences to see if there is any intersection with the alternate unit. We found intersection for the alternate units les paul, gibson les, llyod, and adolph rickenbaker but not for leo. Hence leo get a score of zero and we process other alternative units further. In the next step, we find intersection between content words or topic unit and matched sentences for alternate units found in the previous step. We count no of words that are overlapped and rank alternate units according to this count value. Count of the content word overlap is given below.

3 les paul  
1 gibson les  
0 llyod  
0 leo  
0 adolph rickenbacker

It shows that we have 3 content word intersection for les paul, 1 word intersection for gibson les and 0 for all other alternate units. This ranking is the final truth or confidence ranking for all alternate units.

## 5 Alternative methods that are explored

For disambiguation among the alternative statements, Wikipedia is generally used as an authoritative source. On the other hand DBpedia, Freebase have organized the massive amount of data in a searchable fashion e.g. DBpedia uses SPARQL endpoint, Freebase uses MQL api. Open source implementation of Python wrappers exist for both the interfaces exist and appear to be mature enough for our needs.

### 5.1 Freebase

Freebase has information about approximately 20 million Topics, each one having a unique Id, which can help distinguish multiple entities which have similar names, such as Henry Ford the industrialist vs Henry Ford the footballer. Most of the topics are associated with one or more types[1] (such as people, places, books, films, etc) and may have additional properties like "date of birth" for a person or latitude and longitude for a location. Freebase not only contains data from the Wikipedia but also other sources; users can submit data to the Freebase datastore and expand it in richness. We tinkered with the api[2] and it appeared to be the most viable starting point for the project.

Listing 1: Minimal code to Freebase

```
import freebase
import pprint

query = [{
    "a:starring": [{
        "actor": "Meg Ryan"
    }],
    "b:starring": [{
        "actor": "Tom Hanks"
    }],
    "type": "/film/film",
    "*": [],
}]
```



```

pp = pprint.PrettyPrinter(indent=4)
result = freebase.mqlread(query)

print "Movie names & their various forms"

for i in result:
    pp.pprint(i["key"])

```

#### Listing 2: Cleaned Output

```

Movie names & their various forms
[  '158982',
   'You$0027ve_Got_Mail ',
   '18171032',
   ...,
   'E-m$0040il-f$00FCr-Dich ',
   'youve-got-mail ']
[  '176489',
   'Joe_Versus_the_Volcano ',
   'Joe_Vs$002E_The_Volcano ',
   'Brain_Cloud ',
   ...,
   '2327353',
   'joe-versus-the-volcano ']
[  '226198',
   'Sleepless_in_Seattle ',
   'Sleepless_In_Seattle ',
   ....
   '169146',
   '106482',
   'Insonnia_d$0027amore ',
   '62812',
   'Schlaflos_in_Seattle ',
   'sleepless-in-seattle '\]

```

The above results show how the three movies starring Tom Hanks and Meg Ryan. When we query Google with Tom Hanks and Meg Ryan, the top result is a page from Answers.com where a user has asked which are the movies where the two actors appear together, and the answer lists these three movies namely - “Joe versus the Volcano”, “Sleepless in Seattle” and “You’ve got Mail”. A quick lookup of the Wikipedia and IMDB pages also confirm the same.

### 5.1.1 Use of Freebase

For all the 50 sentences mentioned in the original paper we tried the default POS tagger that comes with the Natural Language toolkit along with NE chunker, Binary NE Chunker and the IEER NE Chunker. None of the yielded good results. So we used the Illinois named entity extractor from UIUC, which gave comparatively better results primarily because its database is built from various sources like the Wikipedia, Brown Hierarchical Word Clusters etc.

- Correctly tagged 19
- Partly correct 14
- Wrong/no identification 12

Consider one of the following sentences:

*"Tom Hanks was the lead actress in the movie Sleepless in Seattle"*

Tom/NNP Hanks/NNP was/VBD the/DT lead/NN actress/NN in/IN the/DT movie/NN Sleepless/NNP in/IN Seattle/NNP Phrases and Named Entities

PERSON: Tom/NNP PERSON: Hanks/NNP GPE: Seattle/NNP

Content words for this sentence are "Tom Hanks", "lead", "actress", "movie", "Sleepless in Seattle".

While "actress" is a domain in freebase, it does not contain anything yet. So we look into the synset of the word "actress" from wordnet, which includes *"female actor"*.

Now for the noun phrase *"Sleepless in Seattle"*, we can generate "id" for the query as "sleepless in seattle". But this id will be associated with many properties. To select the relevant property we can use synset obtained from the actress. And this synset has *actor*, which is one of the property for id *"Sleepless in Seattle"*. Hence the Freebase query can be following:

```
[{
  "id" : "/en/sleepless_in_seattle"
  "/film/film/starring" : [{ "actor" : null }]
}]
```

Now the result of this query returns following output:

```
"code": "/api/status/ok",
"result": [{
  "/film/film/starring": [
    {
      "actor": "Tom Hanks"
    },
  ],
}]
```

```

    {
      "actor": "Meg Ryan"
    },
    {
      "actor": "Bill Pullman"
    },
    {
      "actor": "Rosie O'Donnell"
    },
    {
      "actor": "Rob Reiner"
    },
    {
      "actor": "Victor Garber"
    },
    {
      "actor": "Gaby Hoffmann"
    },
    {
      "actor": "Carey Lowell"
    },
    {
      "actor": "David Hyde Pierce"
    },
    {
      "actor": "Ross Malinger"
    },
    {
      "actor": "Frances Conroy"
    },
    {
      "actor": "Rita Wilson"
    }
  ],
  "id": "/en/sleepless_in_Seattle"
}],
"status": "200 OK",
"transaction_id": "cache;cache03.p01.sjc1:8101;2011-03-09T05:09:44Z;0032"
}

```

One of the actors in this result set is "Tom Hanks" that matches with our content word "Tom Hanks" in the given sentence. Now as we know earlier that actress means female actor, we can use the keyword female to find the fact that female is type of gender and Freebase id of "tom hanks" has a property gender associated with it. So we can formulate following query.

```

[ {
  "id" : "/en/tom_hanks"

```

```
"/people/person/gender" : {}  
}]
```

The result of this query is following:

```
{  
  "code": "/api/status/ok",  
  "result": [{  
    "/people/person/gender": {  
      "id": "/en/male",  
      "name": "Male",  
      .  
      .  
      .  
    }  
  ]  
}
```

Here the gender is Male, which contradicts with our gender female. Hence we can decide that this statement is false.

For finding the true statement i.e. the actress we can use all actors obtained in the first query result and form second query with their names and output the truthful sentence if we get female as the gender.

## 5.2 Dbpedia

DBpedia is a similar project to Freebase, but it focuses mainly on the content available from Wikipedia. It scores in being precisely importing the data from the info boxes in Wikipedia pages, but at this stage it does not seem to be offering anything additional over Freebase [4]. We are yet to explore its programmatic interface [3].

## 5.3 Yago

YAGO is a semantic knowledge base with over 900,000 entities (like persons, organizations, cities, etc.) and uses Wikipedia and Wordnet as its main source of information. We are yet to explore the programmatic interfaces it provides and how we can use it for the project.

## 5.4 Building queries from the data supplied by the user

Formulating a proper Freebase query is for our specific purpose is a different process than the standard way of querying a search engine that does full text search on text documents. We start by introducing the various abstraction levels associated with the freebase data.

- A *type* is a conceptual container of related *properties* commonly needed to describe a certain aspect of a *topic*.
- A *topic* can be assigned one or more types (the default type being /common/topic)
- As *properties* are grouped into *types*, *types* are grouped into *domains*.
- *Domains*, *types*, and *properties* are given IDs in a *namespace/key* hierarchy.
- Common well-known topics are given IDs in the /en namespace, which are human-readable English strings.
- *Topics* are uniquely identified within Freebase by *GUIDs*.
- *Properties* are *multi-value* by default, and multi-value properties and single-value properties can be queried in the same way.

In order to transform a sentence to a freebase query we have to one to one map a content word from the sentence(*TU* plus *DU*) to the above mentioned abstraction. In other words, the process involves identifying the contextual meaning of the content words. Although this is pretty intuitive when we do it manually, trying to achieving this programmatically is one the challenging aspect of the project. One way of doing it is using a part of speech taggers, along with chunk extraction and named entity recognition. (details in next section)

The key point of distinction is that this result set is the set of records from a hierarchical database. We can not stuff in every word from the topic units into a query to freebase, as MQL(metaweb query language) is Query By Example language, and has a rigid structure which is not immediately obvious given a sentence in natural language. We incrementally build a query  $Q$  starting from with one word  $w_i$  from the word list  $L$  extracted from the  $TU$ . Let the results associated with  $q_{w_i}$  be  $(R_{w_i})$ . Initially  $Q = \{q_1 = w_i\}$

The following are the possible cases

- No results - In this case we get the synset from Wordnet  $S_{w_i}$  and repeat the search with each word in the synset  $w_j^s$ .
- If there is no match with the word or its synset, we reject  $w_i$  from the query  $Q$  and move on to the next word in  $L$  and repeat the process.
- If  $R$  is not empty, we retain the  $w_i$  or  $w_j^s$  in  $Q$ . Then we take the each of the remaining words  $w_j$  from  $L$  and  $S_{w_j}$  and search in  $R_{w_i}$ . If  $w_j$  or a synonym of it  $w_j^s$  is found to occur in the result, we augment  $Q$  with  $w_j$  (or  $w_j^s$ ). So  $Q$  now becomes  $Q = \{q_1 = w_i, q_2 = w_j\}$  or  $\{q_1 = w_i, q_2 = w_j^s\}$
- With the new  $Q$  we again query Freebase and repeat the above steps.
- We terminate when all the words (and in their synsets) in  $L$  have been substituted. This allows us to form the most appropriate query  $Q$  from the  $TU$ . Note though this is essentially a breadth first search of the graph, we would not be traversing very deep (though the branching factor can be pretty high) because of the small number of content words in  $TU$  and their synsets.

- If the result returned by this query  $Q$  contains the  $DU$ , we can say with a good degree of confidence that the statement is true.

## 5.5 Wikipedia API

Contrary to our prior report, Wikipedia does have a very rich API, which the wiki software, Mediawiki provides. Of the various features that this API provides there are two searching mechanisms.

### 5.5.1 Opensearch Protocol

The first one is Opensearch protocol, which gets pages whose name case-insensitively match a given string. When default limit(10) is reached, results are ordered by number of incoming links. For example if we search for Forrest Gump, we will get:

Listing 3: Result using Opensearch

```
* "Forrest Gump"
  o "Forrest Gump"
  o "Forrest Gump (character)"
  o "Forrest Gump (novel)"
  o "Forrest Gump (soundtrack)"
  o "Forrest Gump      Original Motion Picture Score"
  o "Forrest Gump (disambiguation)"
```

This to a certain extent provides some semantic information attached alongwith the term.

### 5.5.2 Fulltext search

This provides a richer resultset, which syntactically resembles as the results returned for query on a web search engine. However, the interpretation of the returned results and the methods to analyse it has to significantly different. A real world example might make this distinction more clear. Searching the web is more like asking which books in the library has information on a particular topic, but full text search on wikipedia is more like pulling out a single volume of one's favourite encyclopedia and searching the index pages to find if there is an article on this topic. So each article has a definite focus and it is closely defined by its title. Majority of the wikipedia pages adhere to some structure, which might reveal more than the text that is in there for the article. For example the interwiki links, the See also links etc. point towards related topics. Before exploring those areas, we first focus on the various aspects that are immediately available from the api:

1. srinfo What metadata to return. Type: one of totalhits, suggestion

2. srlimit How many total pages to return. Type: limit
3. srnamespace The namespace(s) to enumerate. Type: namespace
4. sroffset Use this value to continue paging (return by query). Type: integer
5. srprop What properties to return:
  - (a) size - Adds the size of the page in bytes
  - (b) wordcount - Adds the word count of the page
  - (c) timestamp - Adds the timestamp of when the page was last edited
  - (d) score - Adds the score (if any) from the search engine
  - (e) snippet - Adds a parsed snippet of the page
  - (f) titlesnippet - Adds a parsed snippet of the page title
  - (g) redirectsnippet - Adds a parsed snippet of the redirect
  - (h) redirecttitle - Adds a parsed snippet of the redirect title
  - (i) sectionsnippet - Adds a parsed snippet of the matching section
  - (j) sectiontitle - Adds a parsed snippet of the matching section title
  - (k) hasrelated - Indicates whether a related search is available
6. srredirects Include redirect pages in the search. Type: bool
7. srsearch (required) Search for all page titles (or content) that has this value. Type: string
8. srwhat Search inside the text. Searching titles is disabled in Wikipedia.

## 5.6 Problem transformation - Recognizing Text Entailment

While a thorough experimentation needs to be done in how to utilize all these parameters to do a better extraction of the underlying semantics, we realize limiting the knowledge base to Wikipedia, reduces the problem to an instance of Recognizing Text Entailment (RTE). Over the last few years amazing progress has been made in this track, and the state of the art algorithms are quite complicated to be implemented within this short time frame. We start off with a simple idea for RTE, and list some experimentation that we plan to do ahead.

The main problem of RTE is that given a Hypothesis(H) and Text(T), the algorithm has to detect if T entails H.

T:The sale was made to pay Yukos' US\$ 27.5 billion tax bill, Yuganskneftegaz was originally sold for US\$ 9.4 billion to a little known company Baikalfinansgroup which was later bought by the Russian state-owned oil company Rosneft .

H:Baikalfinansgroup was sold to Rosneft.

The above example is taken from Recognizing Textual Entailment (RTE) 3 Challenge Corpora. The correct answer in this case, (i.e. yes T entails H) is easy for humans to comprehend, but obviously the greatest challenging task to do programmatically.

The basis of the transformation is that each *DS* now becomes a Hypothesis(H) and the wikipedia article the Text(T).

### 5.6.1 With or without the doubt unit

The original approach left out the doubt unit in order to not bias the returned results. However, in this case it might actually be beneficial. We can also go the other way, that is using the *DU* to search article and then check in that article if there is sufficient intersection with the remainder of the doubtful statement. While this may work well for names and places, this would not work out for dates enforcing us to have separate rules attached to different data types. We have only tried without the DU right now.

### 5.6.2 Without DU

We remove the doubt unit (DU) and stop words from the topic unit and use the remaining key words to do a full text search. The results returned by Lucene, search engine that powers these searches are very accurate. But our task is much more complex, and it requires even finer analysis of the result.

Of the various properties that a returned result may be decorated with, the two most important ones are:

- *titlesnippet* - This is probably the most important category of all. If the doubt unit is located here then it would mean that there is a wikipedia page for the doubt unit, and the text contains keywords from the doubtful statement. This is the strongest indication that that the statement might be true.
- *snippet* - This is the SRR in this case, that is formed by the concatenation of parts from three sentences that surround matched query keywords in the article content. The important thing to consider over here are the distances between two such sentence parts and the number of matches each part includes. For a long page, there may be located at two very different areas in the page and be about very different contexts. In order to handle this we can take a weighted distance measure that takes the number of words between two matching sentence parts  $0.33 * \text{number of matches in a sentence part} / (\text{length of the sentence part} * \text{length of document})$

Listing 4: Feature extraction initial steps

```
snippet_split_weights = {}
snippet_splits = split snippets by '<b>...</b>'
for snippet_split in snippet_splits:
    snippet_split_len = length(snippet_split)
    snippet_split_match_count = count number of matches in snippet_split
```



```

        snippet_split_weights[snippet_splits.index(snippet_split)]
            = snippet_split_match_count/snippet_split_len
snippet_weight = 0
for snippet_split_weight in snippet_split_weights.values():
    snippet_weight += snippet_split_weight
snippet_weight= 0.33*snippet_weight

#TITLE
title_match_count = count all matches in result['titlesnippet']
title_len = length(result['title'])
title_weight = title_match_count/title_len

#TODO need to determine appropriate wieghts
overall_result_weight = (title_weight+snippet_weight)/2

if regex_tokenize:
    tokenizer = RegexpTokenizer('([A-Z]\.)+|\w+|\$[\d\.]+')

    snippet_tokens = set(tokenizer.tokenize(snippet))
    title_tokens = set(tokenizer.tokenize(result['title']))
    doubt_unit_tokens = set(tokenizer.tokenize(doubt_unit))
else:
    snippet_tokens = set(lower case the snippet and tokenize by space)
    title_tokens = set(lower case the title and tokenize by space)
    doubt_unit_tokens = set(lower case the doubt_unit and tokenize by space)

found = False

#If the match is found in the snippet
if snippet_tokens.intersection(doubt_unit_tokens):
    print "S",';',title_weight, ';', snippet_weight, ';', overall_result_weight
    found = True

#if the match is found in the Title
if title_tokens.intersection(doubt_unit_tokens):
    print "T",';',title_weight, ';', snippet_weight, ';', overall_result_weight
    found = True

#If no match is found
if title_tokens.intersection(doubt_unit_tokens):
    print "N",';',title_weight, ';', snippet_weight, ';', overall_result_weight
    found = True

```

As for the rest of the properties like redirecttitles, redirectsnippets, sectiontitle, sectionsnippets did not return values that seem to be of immediate help. However, they made find their use in

generating alternative units. We have not yet experimented with them yet.

Currently the code that we have written only checks for presence of the doubt unit in each of the above mentioned property, for the fifty sentences that were there in the original paper.

The following are the immediate concerns

- Better matching: We tried two ways to match the DU-s in the returned results, one a conservative approach that would try to tokenize so that abbreviations like "U.S.A" and monetary amounts like "\$23.00" are kept as tokens. In the more relaxed approach the tokens were all lowercased and split by spaces. This obviously is the most crucial part of the entire process and it appears we should have separate strategy based on the data type of the doubt unit.
- Building the classifier: We got the data from TREC 9, which is in a question answer format and have finished converting 100 of the 711 sentences all to statement form with correct answers. Once we have some more sentences labeled we will use it to train a classifier from the features extracted above.

## 5.7 Use of answer.com

We found that answer.com also contains the correct answers to the for the doubt sentences in almost of all the cases. The precise Q&A format of closely matching the sentences excites us about the possibilities of using the above mentioned strategies for this site. A lot of domain specific sites (like stackexchange.com, quora.com) have become very popular since 2009, which deviate from the traditional forums in the sense that the user generated content is voted, summarised and exposed to a programmatic interface.

## References

- [1] <http://blog.freebase.com/2007/03/19/the-type-system-in-freebase/>.
- [2] <http://code.google.com/p/freebase-python/>.
- [3] <http://sourceforge.net/projects/sparql-wrapper/files/sparql-wrapper-python/1.4.2/>.
- [4] <http://wiki.freebase.com/wiki/dbpedia>.
- [5] Clement Yu Xian Li, Weiyi Meng. T-verifier: Verifying truthfulness of fact statements. In *ICDE*, 2011.