A Report

On

Automatic Question Answering System (Revised)

Prepared By

Aditya Sarma A.S.	2013A7PS079P
Sanjay Reddy S.	2013A7PS189P
Vamsi Tadikonda	2013A7PS039P

For the partial fulfilment of the course

Artificial Intelligence (CS F407)

Birla Institute of Technology and Science

Pilani

Rajasthan-333031

Contents

INTRODUCTION

Mankind as a species has always been a different one. They don't just simply survive but create and discover. This is because of their questioning nature. Since time immemorial humans have been asking various questions (in various fields) and tried to find answers to them. This spirit of questioning in this information age is more relevant than ever.

Humans now a days depend mostly on technological advancements like the digitalization. We have started converting everything in print to the digital form. In the early 21st century, Search engines and internet have made the work of humans to collect information very easy. We can access the internet from any place and at any time and search for anything on a search engine. This is in many ways very useful and there is a vast research going on it. But the journey doesn't stop there. Although search engines are giving us a lot of information, it is still not catering to 'individual needs'. The results are too many and are too generic while users want 'specialised' results to their queries. There are many projects going on trying to solve this problem. Question-Answering system is one of them.

MOTIVATION OF THE PROJECT

Question Answering Systems are not the same as a Search Engine. Search Engine answers the query by giving web pages as the result.so we manually have to do the searching for the information in the pages/ documents it returns. Question Answering System helps us in getting a specific answer to a question posed and is more user interactive. To build a question answering system, we would need to explore many fields of artificial Intelligence and computer science. With the interest of automation getting more demand, Question Answering Systems are used a lot in medical science, customer care services, chat bots etc.

As we all know “Necessity is the mother of invention” so this system will be tool to improve our daily lives and we wish to improve the efficiency of how these systems work.

CHALLENGES IN QUESTION ANSWERING

Though Designing a Question Answering System may look simple, there are lot of thing that we need to keep in mind while designing the method and framework of the system. Here a few problems that we need to handle

1. Type of Question
2. Question form
3. Context of Question
4. Data sources
5. Answer extraction
6. Answer formulation
7. Question answering in real time
8. Tackling different Languages
9. Interactive
10. Reasoning Capabilities
11. Information clustering
12. User Information

1. Type of Query: A question may belong to different category and different field of education and depending on its category we require different strategies to answer the question. The implementation might change for each category and the way to interpret and process it also changes. The method to follow changes as the requirement changes. For some of the questions we can simply return the answer directly from the source while for others we may need a deep understanding of the subject and quite possibly include few inductive skills.

Example: Questions like “Who is Immanuel Kant?” may not need more facts when compared to questions like “Why is the sky blue in colour?” Here we need to understand not only facts but we must have knowledge of Scattering of Light.

2 .Question Form: A Question can be asked in different forms like interrogative or the assertive form. The form changes when we ask the question in a different way. To handle these, we need to know what the query is asking for before answering the question itself.

Example: Interrogative form of Question: “Who is Nicki Minaj?”
Assertive form can be “Tell me who Nicki Minaj is?”

3. Context of the Query

Sometimes, questions can be related to the context. The main challenge is to find the user’s intent and not simply answer questions in a “one shoe fits all” mentality. We have to know the user’s context and preferences so as to answer his queries with more precision.
Example: The Question “Where is Taj?” varies from context to context. For a person outside the country it may refer to the location of Taj Mahal in Agra. For a person who is in Mumbai it may be about Taj Hotel.

4 Dynamic Data Sources

Before we can start answering questions we need a data source which will act as the base for the information required for answering the questions. It may be collection of documents or the internet or a database where we can search for. The challenge is to develop an architecture for information access that can ensure freshness and coverage of information in a rapidly growing world. Our data should be stored in such a fashion that easy updating and insertion may be possible in the future. We can use web crawling techniques to extract information from websites. But we must carefully evaluate this data also so that we don’t pass any incorrect information to the user.

5 Answer Extraction

Depending on the query asked we require different information from the data sources. Like we may need to extract a Name from a Question like “Who is the president of India?” and extraction of time or date for a Question like “When was mahatma gandhi born?”

6 Answer Formulation

The code should be written in such a format that very few constraints should be put on the user. Specifically no constraints should be put on the format the user can give into the system. It should be able to understand the intent of the user’s question and reply as accurately as possible irrespective of the structure of his query. In extreme cases it should even try to understand the words in the question even if typed incorrectly. We also might have to generate the final output in a specialised way so that the answers look natural.

7 Question answering in Real Time

This is one of the main concerns especially if we have large databanks. These answering systems are usually slow. Not only should we able to answer the queries in real time but also create/update such databanks as quickly as possible. This later part is especially important if we are using any web-crawler techniques to extract relevant information from Wikipedia or other web related information sites. We can use indexing techniques for faster retrieval but even creating appropriate indexes for the data is in itself a challenge.

8 Multilingual

Though English is the most widely spoken language it doesn’t cater everyone’s needs. It is now being guessed that more than half of existing html web pages have information not in English. Therefore accessing and returning back information in a wide variety of languages is very important for many users. There are many data sources in English. So we translate a given query to a different language and get the answer which is translated back to original language.

Example: “इंडोनेशिया के राष्ट्रपशि कौन है?” is a question in hindi we translate the question to English as “Who is the president of Indonesia?” and find the answer in english and then translate it back to hindi.

9 Interactive

People like a system that is more interactive and user friendly. Instead of traditional systems which answer one question at a time we could allow the system to have a dialogue with the user. This can allow the user to ask very detailed questions on very specific topics. The system can also tell the user that no answer is possible in few questions or ask for more parameters before answering the question, if his question is ambiguous.

Example: if user asks a question like "Tell me a good eatery nearby?" the system should be able to give a response like "Veg or Non Veg?"

10 Reasoning Capabilities

We do not want the system to just give the answer by no logical understanding. It should try to reason out and use inductive reasoning (atleast to the extent of how 'Prolog' does it). It should be able to give answers to non-factoid questions which involve reasoning and logical understanding.

Example: if I give a few statements like "Vamsi is Aditya's brother. Brother of father is uncle. Vamsi is father of Sanjay." and ask a question like "Who is uncle of Sanjay?" Then it should reason and give an answer.

11. Information Clustering

For efficient search results we should cluster the data we have and then ask questions in the specific clusters. This will reduce the time taken besides increasing the reliability of the code. The clusters can be divided based on many things like 'factoid and causal' based or on the type of the question words.

12. User Information

We can give better results if we know information about the user so that the search results can be narrowed down to the required thing the user wants. This can be done factors like the user's location search history, preferences etc.

Example: for a question like "what is interesting in Spain?" the answer changes from person to person. If he likes food we can suggest him some restaurants and if he likes sports then we can show some football matches.

DATASETS

1) The TREC Corpus

TREC is a supports research for the information retrieval community and also provides large data collections for testing their systems with it. It is cosponsored by the National Institute of Standards and Technology (NIST). The TREC dataset is not specific to any particular application, and is independent of the interfaces or optimized response time for searching. It is helpful in designing or testing a retrieval system for some specific users. The TREC provides datasets f or various types of systems. For different categories we have different tracks like Blog, Enterprise, entity etc. It is assumed that the users have the ability to do both high precision and high recall searches, and are will look at many documents and repeatedly modify questions get a higher recall.

2) Wikipedia Dump

Wikipedia provides all available content to users for free in a downloadable form. These databases can be used for many purposes like offline reading and can also be used as a domain for data collection. We have used the Wikipedia dump as the source of information for document extraction in one of our algorithms. The size of the dump will be around 90 gigabytes when extracted completely. This may not be useful for computers with low computation power.

ALGORITHMS/RESEARCH PAPERS IMPLEMENTED:

MICROSOFT ASKMSR QUESTION ANSWERING SYSTEM - THE N-GRAM METHOD:

The Microsoft N-Gram Method unlike a lot of methods does not concentrate on the linguistic analysis of the questions asked to understand it and find out answers. The most common methods used for linguistic analysis include Parts of Speech Tagging, Parsing, Named Entity Extraction etc. Instead the N-Gram method works on the fact that if we have lots of information with a lot of redundancies then an answer to a given question may be found rather easily without complex linguistic analysis and uses the Web as its data source. The basic architecture of the system can be split into the following modules.

1. Query Reformulation
2. N-Gram Mining
3. N-Gram Filtering
4. N-Gram Tiling

Query Reformulation: When a query is given, a list of possible rewrites of the question are generated which are the likely substrings in the declarative answer to the question. The query reformulation can be done using a variety of different techniques, for example by using parts of speech tagging and then rearranging the words accordingly. These query rewrites are then searched in the web(our information source, can be a database also). Most of the rewrites do not result in any relevant matching web pages i.e. some rewrites have a greater chance of returning relevant pages than the others. So we assign scores to the rewrites as well. A higher score to a rewrite means that the rewrite has a greater chance of returning relevant web pages as compared to a lower score rewrite.

In our implementation we used a very simple method for query reformulation. We assumed the second word of the question to be the verb and extracted the verb and generated rewrites by placing the verb at all possible postions in the inital query. Even the score to the rewrites is given in a straightforward way. The rewrites which contain the main verb(assumed to be the second word) are given a score of 5, while the rewrites which do not contain the main verb(only one rewrite) are given a score of 2. All the information related to query rewrites is stored in an object of type Reformulated_Query.

Finally, a python list containing objects of the type Reformulated_Query is formed. This is used for further searching in the get_10_summary() module.

```
def reformulated_queries(question):
  rewrites = []
  if('?' in question):
        question = question[:-1]
  tokens = nltk.word_tokenize(question)
  verb = tokens[1] # assuming the second word is the main verb

  rewrites.append(Reformulated_Query("\"%s %s\"" % (verb, " ".join(tokens[2:])), Q_WEIGHT_Quotes))
  i=2
  while(i<len(tokens)):
        temp="\"%s %s %s\"" % (" ".join(tokens[2:i+1]), verb, " ".join(tokens[i+1:]))
     reform=Reformulated_Query(temp,Q_WEIGHT_Quotes)
     rewrites.append(reform)
     i+=1

  reform = Reformulated_Query(" ".join(tokens[2:]), Q_WEIGHT_UnQuotes      )
  rewrites.append(reform)
  return rewrites
```

N-Gram Mining: Once the set of rewrites have been generated, each rewrite is sent as a search engine query and sent to the search engine from which page summaries will be retrieved.
We used “Google” as our search engine by making use of the python google API. The top 10 page summaries are collected as html text and stored in a list of strings. Then, the summaries returned are cleaned i.e. all the html tokens and some typical words like {Cached, Similar} are removed and only the text part of the page summary is retrieved from the html summary.

N-Gram Filtering: From the text retrieved from html summary sentences are separated. On each sentence a filtering process is carried out, i.e. all the words in the sentence that also appear in the query are filtered out to get Filtered sentences. From each filtered sentence all possible n-grams are generated. Each n-gram is given a score as per the base score which is the score of the reformulated query multiplied by CAPILTALIZATION_FACTOR to the power of the Number of Capitalized words in the n-gram. This is a very simple method which biases outputs towards Proper nouns. This is not the ideal method. Of course, this method will fail if there is unnecessary capitalization in the text. And also may return totally unrelated words like “The” “Of” etc if they are capitalized.

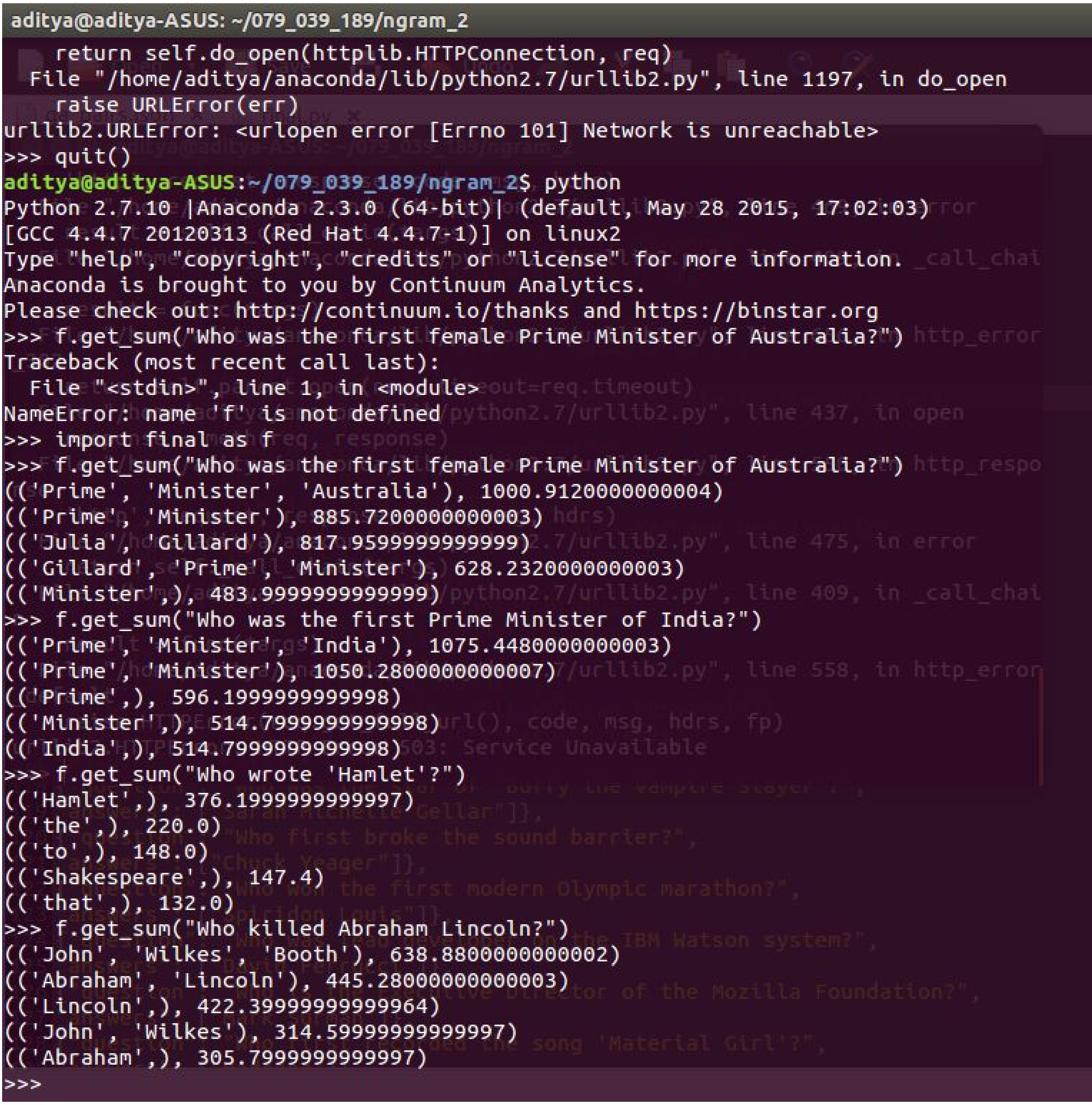
N-Gram Filtering could have been performed in an advanced specialized manner described below.
N-Gram Filtering: Next, the n-grams are filtered and reweighted according to how well each candidate matches the expected answer-type. The system uses filtering in the following manner. The query is analysed and is tagged as one of seven question types, such as who-question, what type of question, or how type of question. Based on the question classification tag that is assigned, the system determines the set of filters to apply to the answers found during the collection of n-grams. The candidate n-grams are then rescored after corresponding post processing.1

N-Gram Tiling: According to the paper, a tiling algorithm also is to be implemented to get really accurate answers. The Tiling algorithm is explained below.

N-Gram Tiling: This algorithms merges similar answers and makes longer answers from overlapping common smaller fragments. This algorithm moves greedily from the highest-scoring candidate to all other candidates (up to a certain threshold). They are checked to find out if tiling can be done on them. If yes, the candidate with the more score is replaced wih the bigger n-gram tiled candidate and the low scoring candidate is removed. The terminating condition of the algorithm is when no further n-grams can be tiled..

Note 1: In our implementation, we are showing the first five most suitable answers according to our algorithm as sometimes we may futile answers as the highest score answer.
Note 2: Our other implementation, FuzzyWuzzy can be used to increase answer accuracy and for pattern matching in the text retrieved from page summaries.

Results:



```
    return self.do_open(httplib.HTTPConnection, req)
  File "/home/aditya/anaconda/lib/python2.7/urllib2.py", line 1197, in do_open
    raise URLError(err)
urllib2.URLError: <urlopen error [Errno 101] Network is unreachable>
>>> quit()
aditya@aditya-ASUS:~/079_039_189/ngram_2$ python
Python 2.7.10 |Anaconda 2.3.0 (64-bit)| (default, May 28 2015, 17:02:03)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://binstar.org
>>> f.get_sum("Who was the first female Prime Minister of Australia?")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'f' is not defined
>>> import final as f
>>> f.get_sum("Who was the first female Prime Minister of Australia?")
(('Prime', 'Minister', 'Australia'), 1000.9120000000004)
(('Prime', 'Minister'), 885.7200000000003)
(('Julia', 'Gillard'), 817.9599999999999)
(('Gillard', 'Prime', 'Minister'), 628.2320000000003)
(('Minister',), 483.9999999999999)
>>> f.get_sum("Who was the first Prime Minister of India?")
(('Prime', 'Minister', 'India'), 1075.4480000000003)
(('Prime', 'Minister'), 1050.2800000000007)
(('Prime',), 596.1999999999998)
(('Minister',), 514.7999999999998)
(('India',), 514.7999999999998)
>>> f.get_sum("Who wrote 'Hamlet'?")
(('Hamlet',), 376.1999999999997)
(('the',), 220.0)
(('to',), 148.0)
(('Shakespeare',), 147.4)
(('that',), 132.0)
>>> f.get_sum("Who killed Abraham Lincoln?")
(('John', 'Wilkes', 'Booth'), 638.8800000000002)
(('Abraham', 'Lincoln'), 445.2800000000003)
(('Lincoln',), 422.39999999999964)
(('John', 'Wilkes'), 314.59999999999997)
(('Abraham',), 305.7999999999997)
>>>
```

Carefully observe the answers to all the questions in the above image. You will come across all the strong-points and drawbacks of our algorithm…

FUZZY WUZZY IMPLEMENTATION:

Fuzzy Wuzzy is a module in python used for comparing strings in an efficient and easy way. We use it in our system for comparing user's query with each entry in the database.

Any Question Answering system consists of two main parts: Extracting data from a source and Searching through it in reference to the given query. These parts are independent from each other (They can be combined to give better results however). Fuzzy Wuzzy deals with solving the second part. So combining google extraction (as described in N-Gram Method) with this module is a very viable option. But for simplicity we have used a hard-coded database in the file.

Database:

- We have segregated the questions into various lists by Question Words like 'Who', 'How', 'What'.
- The questions are taken from the TREC database and also from START project of MIT.
- Each entry of the list is a tuple ("processed string","answer")
- During the processing of queries we remove various frequent words like 'is', 'of', 'the' etc. in order to improve accuracy and execution time.
- Using the function 'structure' we can add more questions to the local question bank. It first searches for the Question Word in the sentence and adds to the respective list after some processing (removal of common words)

Query:

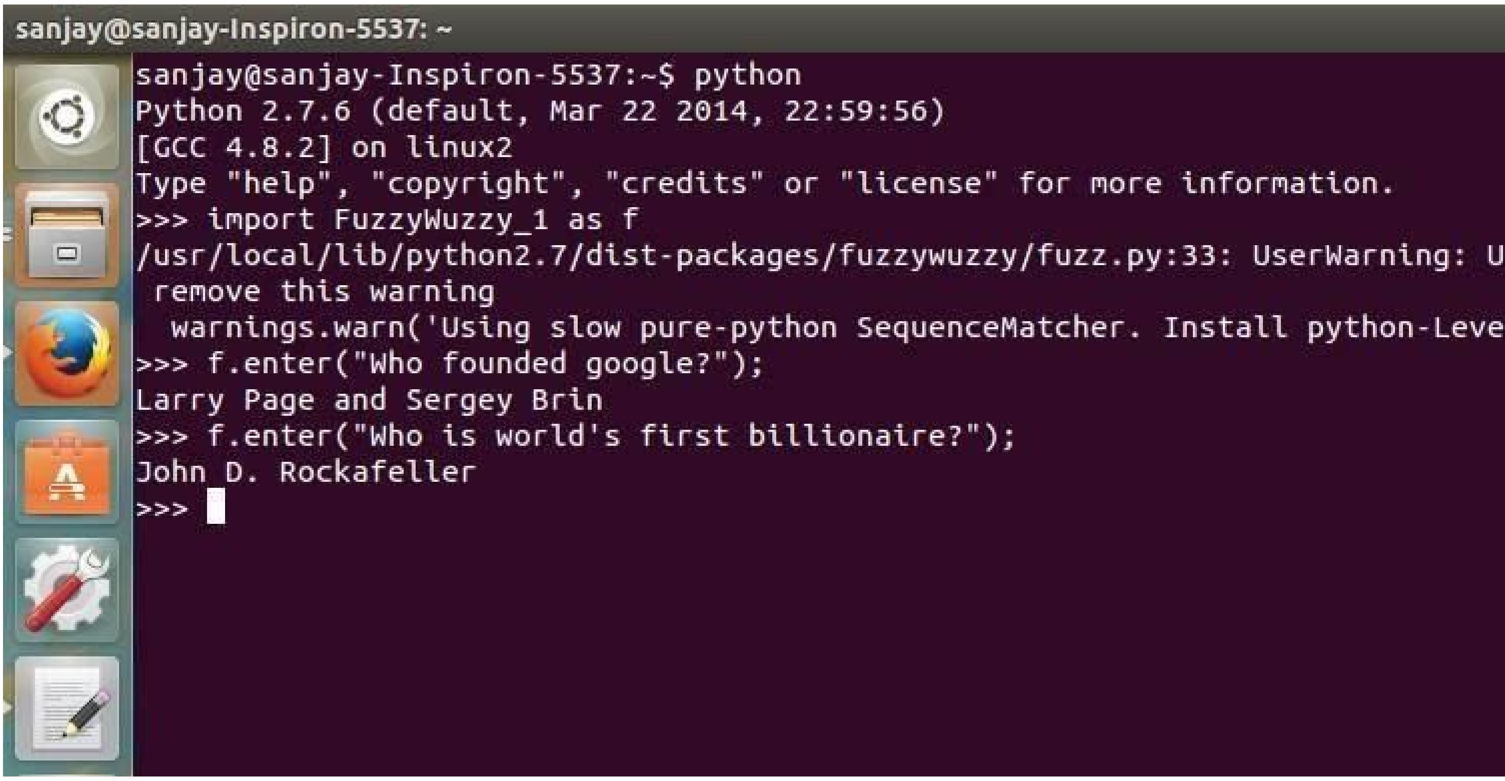
- The function 'enter' accepts the user query from terminal. It processes it and finds out the Question Word in it.
- It then iterates over each question in that list and using fuzzy wuzzy finds out the closest match.
- Fuzzy Wuzzy is a string comparison mainly using levenshtein distance.

>>> fuzz.token_sort_ratio("wuzzy was a rat", "wuzzy wuzzy was a cat")

Result = 78

- Then it simply returns the answer (2nd element of the tuple) of the closest match.

Result:



```
sanjay@sanjay-Inspiron-5537:~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import FuzzyWuzzy_1 as f
/usr/local/lib/python2.7/dist-packages/fuzzywuzzy/fuzz.py:33: UserWarning: U
 remove this warning
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-Leve
>>> f.enter("Who founded google?");
Larry Page and Sergey Brin
>>> f.enter("Who is world's first billionaire?");
John D. Rockafeller
>>> 
```

CAUSAL TYPE QUESTIONS: “WHY” TYPE QUESTIONS.

This method is a specialized method for handling only why type questions unlike N-Gram which handles all types of queries although it performs well for questions which have proper nouns as answers.

This method used the Offline Wikipedia XML dump at dataset and uses a WikiExtractor module for answer extraction. It has an interactive environment. Uses a tornado web server. Training is done on the wikipedia xml dump and then questions are asked to which answers are extracted.

Development Environment:

- We have used Python 2.7 as our programming language and tested the system in Ubuntu (some libraries/modules can't be installed in Windows)
- For Fuzzy-Wuzzy method we have to include the nltk library (for parsing) (http://www.nltk.org/install.html) and fuzzy wuzzy module (for string matching) (https://pypi.python.org/pypi/fuzzywuzzy)
- For N-Gram the following libraries are required:

1. BeautifulSoup (for extracting html files from web) (http://www.crummy.com/software/BeautifulSoup/bs4/doc/)
2. pickle (for operations performed on objects) (https://docs.python.org/2/library/pickle.html)
3. We have used Google.py for searching google ()

PERFORMANCE EVALUATION:

Fuzzy-Wuzzy

1) This approach is like a lazy learner in machine learning (Best analog is K-NN algorithm). It is very dependent on test data and prepares a local model to answer, when a query is given to it by observing the sentence closest to the query.

2) **Advantages:**

· This gives almost 100% correct results if data is present. Also if data is already collected then execution time is only comparing strings and hence this method will give much faster results when compared with other methods. Time complexity is therefore only that of string matching and is much lower when compared to other methods.

· Any Question-Answering system has two parts: Retrieval of information and comparing the given query with collected data. Fuzzy-Wuzzy module is a way of this comparison. So this method can actually be complimented with other implementations to give much better results.

· The writing of this code is very intuitive and can very easily be interfaced with other methods if required.

· This method is capable in handling grammar variations. This means that the user can use any sentence correction he wants but the system still gives the correct answer (since it is pattern matching).

3) **Disadvantages:**

· In our case we have used a local database and for it to yield proper results, we need huge amount of data. Although having big data is a problem in almost all of the approaches, it is particularly important here since in other cases we have local data corresponding to that 'particular query' only but here we must keep 'all' the data locally in the memory in anticipation of any query. Space complexity is very high.

· Although grammatical variations are handled well, different words in query still can lead to incorrect results. To give correct responses even to those, the only option here is to include those variations also in the database which further increases the data problem and also causes redundancy. A better alternative is to use 'word-net' for getting synonyms of the words in query.

4) **Time complexity (approximate):**

· The time complexity for comparing strings is O(length1xlength2)

· It performs linear search to go through the whole dataset one by one.

5) **Testing:**

· This method gives 100% accuracy if the data is present but 0% if not present in the local database.

No. of Questions	Correct Answers	Wrong answers
100	100	0

N-gram

1) This is also a lazy learner but handles few kinds of variations much better.

2) **Advantages:**

· This uses google data for search and handles variations involving different words much better since the repository is big. Also because of the same reason there is no need to update the data as in the previous case.

· As mentioned before the space complexity is smaller (in comparison) because it stores data corresponding to that query only.

· This method gives better results for bigger length questions as it increases the accuracy of the top 10 google results.

3) **Disadvantages:**

· Since it is extracting top 10 results from google, the time complexity is huge. Also the production of various combinations (called 'carpet-bombing') of the given sentence is compute-intensive and yet many of the possibilities are incorrect. We can use language parsers to improve this, but they further increase execution time.

· Here the placement of verbs in the sentence is very important (grammatical variation is less). In our code we simply assume the second word as the main verb, which is a common case but not the only case.

4) **Time Complexity (approximate):** Product of

· Length of string

· Search time in google (This takes seconds)

· No. of summaries (usually 10)

· Avg. no of sentences in the summary

· $({}^{n}C_{1}+{}^{n}C_{2}+{}^{n}C_{3})$ where

'n' is avg. no. of words per sentence

5) **Testing:**

· Although this method gave right answers, google actually prevents web crawling so we couldn't try our data set. Still the few questions we entered gave us good responses.

For 'Why' type Questions

1) Advantages: Very practical since wikipedia is a widely used information source

2) Disadvantages:

· Offline Wikipedia is required in an xml format along with the program.

3) Testing:

· We unfortunately could not test this method since the offline xml file was of 80 GB. But the implementation seemed correct. The accuracy still should be high.

REFERENCES:

1. Brill, Eric, Susan Dumais, and Michele Banko. "An Analysis of the AskMSR Question-answering System." *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - EMNLP '02*. Print.
2. Ramakrishnan, Ganesh, Soumen Chakrabarti, Deepa Paranjpe, and Pushpak Bhattacharya. "Is Question Answering an Acquired Skill?"*Proceedings of the 13th Conference on World Wide Web - WWW '04*. Print.
3. Sucunuta, Manuel E., and Guido E. Riofrio. "Architecture of a Question-Answering System for a Specific Repository of Documents." *2010 2nd International Conference on Software Technology and Engineering*. Print.
4. Katz Boris, Borchardt Gary,Felshin Sue. "Syntactic and Semantic Decomposition Strategies for Question Answering from Multiple Resources"
5. S. Quarteroni and S. Manandhar (2009). Designing an interactive open-domain question answering system. Natural Language Engineering, 15, pp 73-95.
6. WordNet - Princeton University "About WordNet." WordNet. Princeton University. 2010. http://wordnet.princeton.edu
7. Fuzzy Wuzzy - Python Documenation - http://docs.python.org
8. The TREC dataset for Question Answering Systems - http://trec.nist.gov/data/qamain.html
9. DBPedia - http://wiki.dbpedia.org/
10. Alicebot - Application of AIML - http://www.alicebot.org/aiml.html