

Ranking of Answers in Question Answering (CQA) platform with the help of Natural Language Toolkit(NLTK) in Python

Report submitted for the fulfilment of the requirements for the degree of Bachelor of Technology in **Information Technology**

Submitted by

Rohit Jha(11700214058)

Angshu Mitra(11700214012)

Supriya Basu Choudhury(11700214078)

Under the Guidance of
Amit Khan
(Assistant Professor, Department of Information Technology)



RCC Institute of Information Technology Canal South Road, Beliaghata,
Kolkata - 700 015 [Affiliated to West Bengal University of Technology]

RCC INSTITUTE OF INFORMATION TECHNOLOGY

KOLKATA – 700015, INDIA



CERTIFICATE

The report of the Project titled Ranking of Answers in Question Answering (CQA) platform with the help of Natural Language Toolkit (NLTK) in Python submitted by (Roll No.: IT2014/023, IT2014/011, IT2014028 of B. Tech. (IT) 8th Semester of 2018) has been prepared under our supervision for the fulfilment of the requirements for B Tech (IT) degree in West Bengal. University of Technology
The report is hereby forwarded.

OPTIONAL IN CASE

Mr. Amit Khan
Assistant Professor(IT)
RCC Institute Of Information Technology

Mr. Amit Khan
Dept. of Information Technology
RCCIIT, Kolkata

Countersigned by

.....
Dr. Abhijit Das
Information Technology
RCC Institute of Information Technology, Kolkata – 700 015, India

ACKNOWLEDGEMENT

We express our sincere gratitude to Mr. Amit Khan of Department of Information Technology, RCCIIT and for extending his valuable times for us to take up this problem as a Project.

We are also indebted to Dr. Indrajit Pan for his unconditional help and inspiration.

Last but not the least we would like to express our gratitude to Dr. Abhijit Das (HOD) of our department who helped us in his own way whenever needed.

Date: 2018

Rohit Jha

Reg. No.:141170110153
Roll No.: 1700214058
B. Tech (IT) – 8th Semester,
Session-2014-2018, RCCIIT

Supriya Basu Chowdhury

Reg.No.:141170110173
Roll No.:11700214078
B. Tech (IT)-8th Semester,
Session-2014-2018, RCCIIT

Angshu Mitra

Reg.No.:141170110107
Roll No.: 11700214012
B. Tech (IT)-8th Semester,
Session-2014-2018, RCCIIT

RCC INSTITUTE OF INFORMATION TECHNOLOGY

KOLKATA – 700015, INDIA



CERTIFICATE of ACCEPTANCE

The report of the Project titled **Ranking of Answers in Question Answering (CQA) platform with the help of Natural Language Toolkit(NLTK) in Python** submitted by Rohit Jha (Roll No.: IT2014/028 of B. Tech. (IT) 8th Semester of 2018), Supriya Basu Chowdhury (Roll No.: IT2014/023 of B. Tech. (IT) 8th Semester of 2018), Angshu Mitra (Roll No.: IT2014/011 of B. Tech. (IT) 8th Semester of 2018) are hereby recommended to be accepted for the partial fulfilment of the requirements for B Tech (IT) degree in West Bengal. University of Technology

Name of the Examiner

Signature with Date

1.....

.....

2.....

.....

3.....

.....

4.....

.....

TABLE OF CONTENTS

Topics NO	Page
1. Introduction	1
2. Problem Analysis	2
3. Review of Literature	4
4. Flow Chart	6
5. Problem Discussion	7
6. Implementation Details	9
7. Sample Output	26
8. Conclusion	28
9. Reference	29
10. Appendix (Program Code)	30

INTRODUCTION:

Within the natural language processing (NLP) community, similarity between texts (text similarity, henceforth) is a ubiquitous notion and utilized in a wide range of tasks such as question answering (Lin and Pantel, 2001), automatic essay grading (Attali and Burstein, 2006), or paraphrase recognition (Dolan et al., 2004). However, text similarity is often used as an umbrella term covering quite different phenomena – as opposed to the notion of similarity in psychology, which is well studied and captured in formal models such as the set-theoretic model (Tversky, 1977) or the geometric model (Widdows, 2004). We argue that the seemingly simple question “How similar are two texts?” cannot be answered independently from asking what properties make them similar. Goodman (1972) gives a good example for physical objects regarding the situation of a baggage check at the airport: While a spectator might compare bags by shape, size, or color, the pilot only focuses on a bag’s weight, and a passenger compares bags by just destination and ownership. Similarly, texts also have particular inherent properties that need to be considered in any attempt to judge their similarity (Bär et al., 2011). Take for example two novels by the famous 19th century Russian writer Leo Tolstoy. A reader may readily argue that these novels are completely dissimilar due to different plots, people, or places. On the other hand, a second reader (e.g. a scholar overseeing texts of disputed authorship) may argue that both texts are indeed highly similar because of their stylistic similarity. In consequence, text similarity remains a loose notion unless we provide a frame of reference. We argue that text similarity cannot be seen as a fixed, axiomatic notion. Rather, we need to define in what way two texts are similar. From a human-centered perspective, we say that text similarity is a function between two texts t_1 and t_2 which can be informally characterized by the readers’ shared view on the text characteristics along which similarity is to be judged. However, to the best of our knowledge the definition of appropriate text characteristics for text similarity computation has not been tackled yet in any previous research. We thus further argue that text similarity can be judged along different text dimensions, i.e. groups of text characteristics which are perceived by humans and for which we provide empirical evidence. For example, a scholar in digital humanities may be less interested in texts that share similar contents – as opposed to e.g. near-duplicate detection (see Section 2) – but may rather be looking for text pairs which are similar with respect to their style and structure. Throughout this work and in particular in Section 3, we will elaborate on the idea of text dimensions and further discuss suitable dimensions for text similarity tasks.

Problem Analysis

Understanding the intent behind a new question is a natural direction for improving CQA services, since it can supply users with more personalized, and more effective CQA services tailored to their information needs. For example, we may want to employ different strategies to answer questions with different intent. However, current research on user intent in search engines cannot be directly applied to CQA services.

In CQA users normally ask natural language questions, which are addressed to humans, whereas in Web search users submit keyword queries which are addressed to computerized algorithms. More specifically, this leads to the following five major differences between CQA questions and search engine queries:

1. Many CQA questions are inherently subjective. It has been shown that the proportion of Yahoo! Answers oriented to factual question answering is decreasing while subjective/complex question answering is gradually increasing .
2. Many CQA questions are socially motivated, as users know that the answers to their questions would be coming from other users in the community. Instead of satisfying an information need, such questions are actually about establishing social connections (e.g., finding a date), or about generating some empathy (e.g., complaining), or just for entertainment purposes (e.g. telling jokes).
3. Even though about 10% of queries submitted to search engines are in question format , they are quite different from the question patterns used in CQA services. For example, instead of using the common question format "What is a", or "Where is" in CQA, question queries in search engines are more likely to be the formats as "I need", "I want", "Show me".
4. CQA questions are more likely to have additional constraints, since they are usually longer and more complex than the search engine queries. For example, people may ask something in a specific area (e.g., looking for restaurants), or within a specific time frame (e.g., seeking for news).

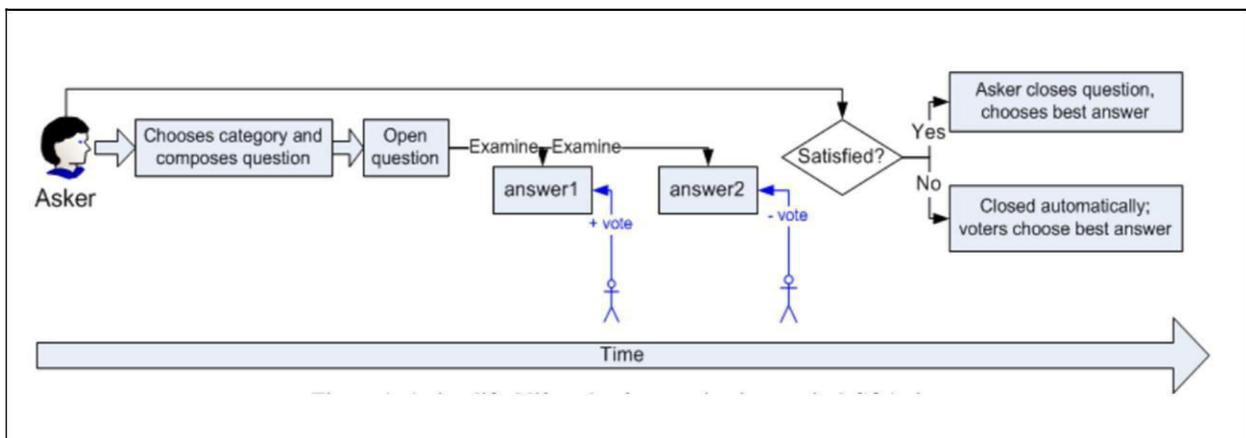
5. Compared with search engines, CQA services have richer information, which can be used to characterize one's social status. For instance, each user has their unique asking and answering history; each question may correspond to a best answer, and an upvote/downvote value; furthermore, some user may have the pattern of asking questions in several specific topics (e.g., Traveling). This kind of rich information can help CQA system to reveal the user intent by providing evidence from the user's perspective, in addition to the surface textual features from the questions themselves.

Furthermore, even though there have been CQA studies, which investigate strategies for one or two dimensions of the user intents, they mostly summarize each question as a clear and simple information need (so that the computer can understand it easily). Question answering systems are required to understand the user intent at a deeper level. In this thesis we investigate potential answers to the following three questions regarding user intent in CQA:

How to categorize different user intents in CQA? (taxonomy)

How to automatically identify the user intents of a question from a CQA service? (classifier) How to incorporate the user intents to improve the performance of CQA services? (e.g., question retrieval and answer validation)

Investigating all these questions form a picture depicting the multi-dimensional nature of the user intent would help us not only to understand the question more deeply but also in a broader context.

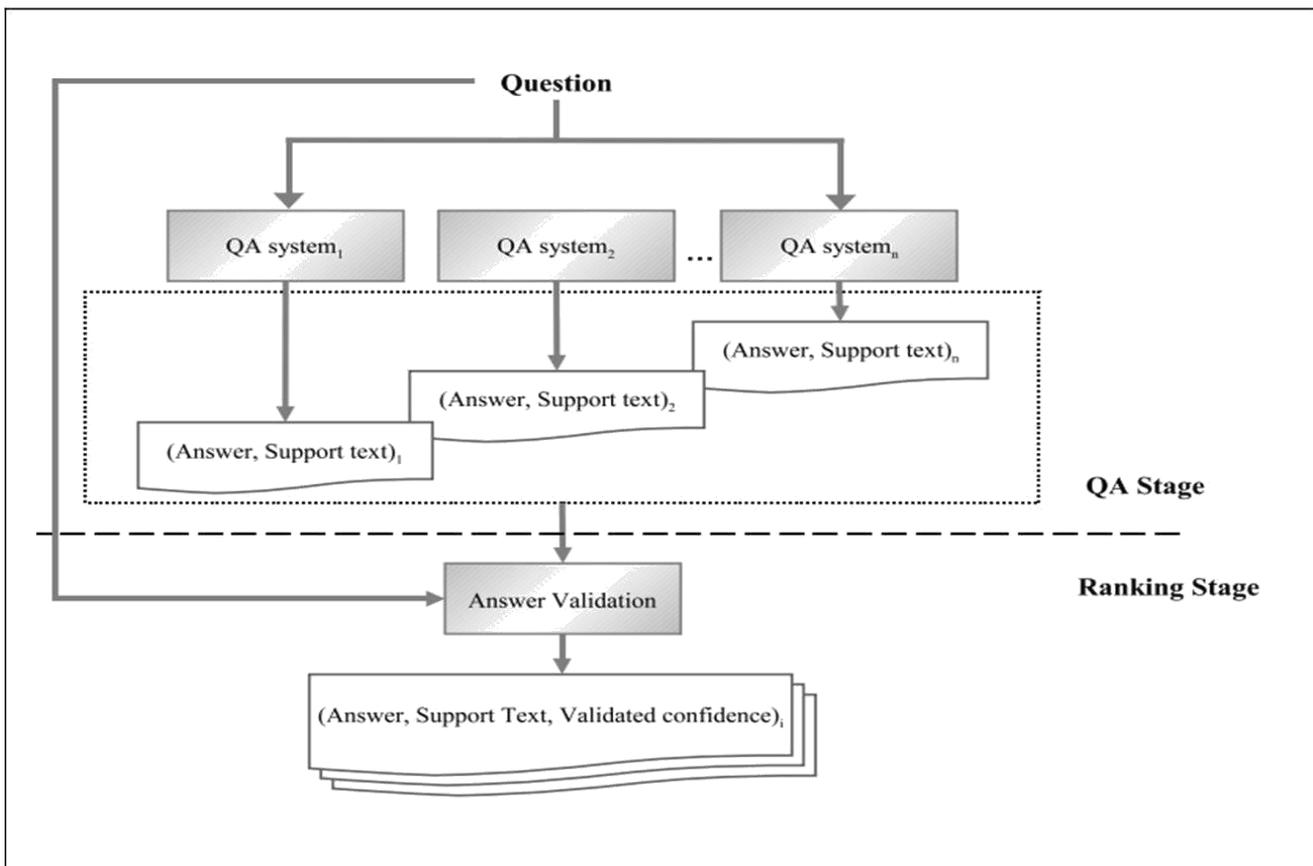


Review of Literature

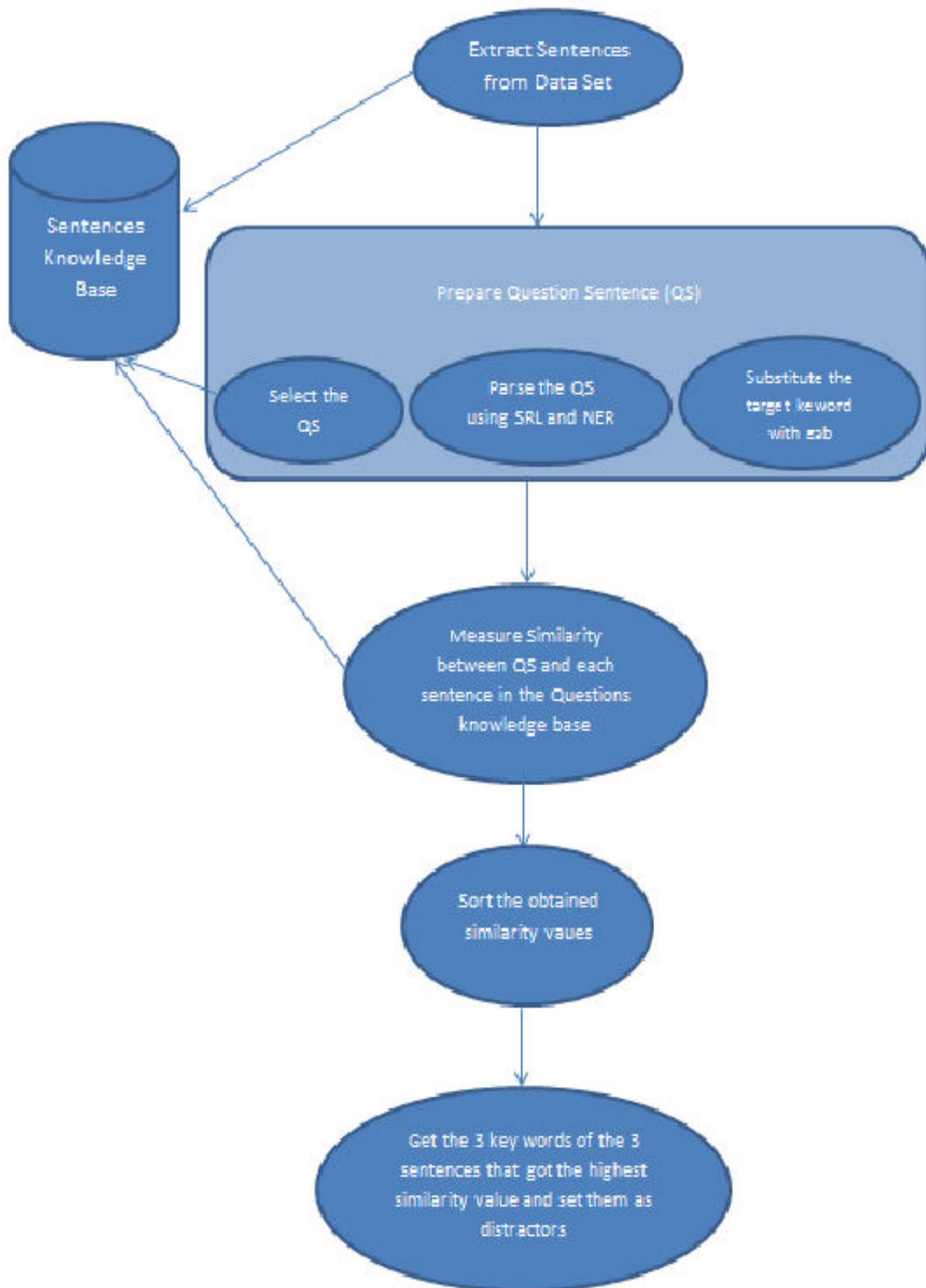
The quest for knowledge is deeply human, and so it is not surprising that practically as soon as there were computers, and certainly as soon as there was natural language processing, we were trying to use computers to answer textual questions. By the early 1960s, there were systems implementing the two major modern paradigms of question answering—IR-based question answering and knowledge-based question answering to answer questions about baseball statistics or scientific facts. Even imaginary computers got into the act. Deep Thought, the computer that Douglas Adams invented in *The Hitchhiker’s Guide to the Galaxy*, managed to answer “the Great Question of Life The Universe and Everything” (the answer was 42, but unfortunately the details of the question were never revealed). More recently, IBM’s Watson question-answering system won the TV gameshow *Jeopardy!* in 2011, beating humans at the task of answering questions like WILLIAM WILKINSON’S “AN ACCOUNT OF THE PRINCIPALITIES OF WALLACHIA AND MOLDOVIA” INSPIRED THIS AUTHOR’S MOST FAMOUS NOVEL¹ Although the goal of quiz shows is entertainment, the technology used to answer these questions both draws on and extends the state of the art in practical question answering, as we will see. Most current question answering systems focus on factoid questions. Factoid questions are questions that can be answered with simple facts expressed in short text answers. The following factoid questions, for example, can be answered with a short string expressing a personal name, temporal expression, or location: (28.1) Who founded Virgin Airlines? (28.2) What is the average age of the onset of autism? (28.3) Where is Apple Computer based? In this chapter we describe the two major modern paradigms to question answering, focusing on their application to factoid questions. The first paradigm is called IR-based question answering or sometimes text based question answering, and relies on the enormous amounts of information available as text on the Web or in specialized collections such as PubMed. Given a user question, information retrieval techniques extract passages directly from these documents, guided by the text of the question. The method processes the question to determine the likely answer type (often a named entity like a person, location, or time), and formulates queries to send to a search engine. The search engine returns ranked documents which are broken up into suitable passages and reranked. Finally candidate answer strings are extracted from the passages and ranked.

In the second paradigm, knowledge-based question answering, we instead build a semantic representation of the query. The meaning of a query can be a full predicate calculus statement. So the question What states border Texas?—taken from the GeoQuery database of questions on U.S. Geography (Zelle and Mooney, 1996)— might have the representation: $\lambda x.state(x) \wedge borders(x, texas)$ Alternatively the meaning of a question could be a single relation between a known and an unknown entity. Thus the representation of the question When was Ada Lovelace born? could be birth-year (Ada Lovelace, ?x). Whatever meaning representation we choose, we’ll be using it to query

databases of facts. These might be complex databases, perhaps of scientific facts or geospatial information, that need powerful logical or SQL queries. Or these might be databases triple stores of simple relations, triple stores like Freebase or DBpedia introduced in Chapter 20. Large practical systems like the DeepQA system in IBM's Watson generally are hybrid systems, using both text datasets and structured knowledge bases to answer questions. DeepQA extracts a wide variety of meanings from the question (parses, relations, named entities, ontological information), and then finds large numbers of candidate answers in both knowledge bases and in textual sources like Wikipedia or newspapers. Each candidate answer is then scored using a wide variety of knowledge sources, such as geospatial databases, temporal reasoning, taxonomical classification, and various textual sources.



Flow Chart



Problem Discussion

We have to measure similarity between various group of texts with the help of various python libraries and toolkit. One of the is the NLTK module which helps in similarities. The NLTK module is a massive tool kit, aimed at helping you with the entire Natural Language Processing (NLP) methodology. NLTK will aid you with everything from splitting sentences from paragraphs, splitting up words, recognizing the part of speech of those words, highlighting the main subjects, and then even with helping your machine to understand what the text is all about. In this series, we're going to tackle the field of opinion mining, or sentiment analysis.

In our path to learning how to do sentiment analysis with NLTK, we're going to learn the following:

- Tokenizing - Splitting sentences and words from the body of text.
- Part of Speech tagging
- Removing stop words from the tokenized words
- Lemmatization of the texts
- How to tie in Scikit-learn (sklearn) with NLTK

In order to get started, you are going to need the NLTK module, as well as Python.

If you do not have Python yet, go to Python.org and download the latest version of Python if you are on Windows. If you are on Mac or Linux, you should be able to run an apt-get install python3

Now that you have all the things that you need, let's knock out some quick vocabulary:

- Corpus - Body of text, singular. Corpora is the plural of this. Example: A collection of medical journals.
- Lexicon - Words and their meanings. Example: English dictionary. For example: To a financial investor, the first meaning for the word "Bull" is someone who is confident about the market, as compared to the common English lexicon, where the first meaning for the word "Bull" is an animal. As such, there is a special lexicon for financial investors, doctors, children, mechanics, and so on.
- Token - Each "entity" that is a part of whatever was split up based on rules. For examples, each word is a token when a sentence is "tokenized" into words. Each sentence can also be a token, if you tokenized the sentences out of a paragraph.

System Requirement Specification

The purpose behind the Software Requirements Specification document is to describe the resources and management of those resources used in the design of the Question Answering System. The System Requirements and Specifications will further provide details regarding the functional and performance related requirements of the algorithm.

Hardware and Software Requirements:

The hardware and software requirements of the system which are required for the implementation of the project in a system.

Software requirements

Technologies : Python
Tools : NLTK 3.0,Scikit,sklearn
Domain : Machin Learning,Natural language Processing
Jdk : Version 1.6 or above
Python : 2.6 or above
Operating system : Windows 8.1

Hardware requirements

Processor : Intel Core Processor
RAM : 3 GB (minimum)
Hard Disk : 80 GB(minimum)

Implementation Details

Since, text is the most unstructured form of all the available data, various types of noise are present in it and the data is not readily analyzable without any pre-processing. The entire process of cleaning and standardization of text, making it noise-free and ready for analysis is known as text preprocessing.

It is predominantly comprised of three steps:

- Noise Removal
- Lexicon Normalization
- Object Standardization

1. Tokenization

These are the words you will most commonly hear upon entering the Natural Language Processing (NLP) space, but there are many more that we will be covering in time. With that, let's show an example of how one might actually tokenize something into tokens with the NLTK module.

```
from nltk.tokenize import word_tokenize
```

```
EXAMPLE_TEXT = "Hello Mr. Smith, how are you doing today? The weather is great,  
and Python is awesome. The sky is pinkish-blue. You shouldn't eat cardboard."  
print(sent_tokenize(EXAMPLE_TEXT))
```

At first, you may think tokenizing by things like words or sentences is a rather trivial enterprise. For many sentences it can be. The first step would be likely doing a simple `.split(' ')`, or splitting by period followed by a space. Then maybe you would bring in some regular expressions to split by period, space, and then a capital letter. The problem is that things like Mr. Smith would cause you trouble, and many other things. Splitting by word is also a challenge, especially when considering things like concatenations like we and are to we're. NLTK is going to go ahead and just save you a ton of time with this seemingly simple, yet very complex, operation.

The above code will output the sentences, split up into a list of sentences, which you can do

things like iterate through with a for loop. ['Hello Mr. Smith, how are you doing today?', 'The weather is great, and Python is awesome.', 'The sky is pinkish-blue.', 'You shouldn't eat cardboard.']

So there, we have created tokens, which are sentences. Let's tokenize by word instead this time:

```
print(word_tokenize(EXAMPLE_TEXT))
```

Now our output is: ['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'weather', 'is', 'great', ',', 'and', 'Python', 'is', 'awesome', ',', 'The', 'sky', 'is', 'pinkish-blue', ',', 'You', 'should', 'n't', 'eat', 'cardboard', '.']

There are a few things to note here. First, notice that punctuation is treated as a separate token. Also, notice the separation of the word "shouldn't" into "should" and "n't." Finally, notice that "pinkish-blue" is indeed treated like the "one word" it was meant to be turned into.

2. Stop words removal

The idea of Natural Language Processing is to do some form of analysis, or processing, where the machine can understand, at least to some level, what the text means, says, or implies.

This is an obviously massive challenge, but there are steps to doing it that anyone can follow. The main idea, however, is that computers simply do not, and will not, ever understand words directly. Humans don't either *shocker*. In humans, memory is broken down into electrical signals in the brain, in the form of neural groups that fire in patterns. There is a lot about the brain that remains unknown, but, the more we break down the human brain to the basic elements, we find out basic the elements really are. Well, it turns out computers store information in a very similar way! We need a way to get as close to that as possible if we're going to mimic how humans read and understand text. Generally, computers use numbers for everything, but we often see directly in programming where we use binary signals (True or False, which directly translate to 1 or 0, which originates directly from either the presence of an electrical signal (True, 1), or not (False, 0)). To do this, we need a way to convert words to values, in numbers, or signal patterns. The process of converting data to something a computer can understand is referred to as "pre-processing."

One of the major forms of pre-processing is going to be filtering out useless data. In natural language processing, useless words (data), are referred to as stop words.

Immediately, we can recognize ourselves that some words carry more meaning than other words. We can also see that some words are just plain useless and are filler words. We use them in the English language, for example, to sort of "fluff" up the sentence so it is not so strange sounding. An example of one of the most common, unofficial, useless words is the phrase "umm." People stuff in "umm" frequently, some more than others. This word means nothing, unless of course we're searching for someone who is maybe lacking confidence, is confused, or hasn't practiced much speaking. We all do it, you can hear me saying "umm" or "uhh" in the videos plenty of ...uh ... times. For most analysis, these words are useless.

We would not want these words taking up space in our database or taking up valuable processing time. As such, we call these words "stop words" because they are useless, and we wish to do nothing with them. Another version of the term "stop words" can be more literal: Words we stop on.

For example, you may wish to completely cease analysis if you detect words that are commonly used sarcastically and stop immediately. Sarcastic words, or phrases are going to vary by lexicon and corpus. For now, we'll be considering stop words as words that just contain no meaning, and we want to remove them.

You can do this easily, by storing a list of words that you consider to be stop words. NLTK starts you off with a bunch of words that they consider to be stop words, you can access it via the NLTK corpus with:

```
from nltk.corpus import stopwords
```

Here is the list:

```
set(stopwords.words('english'))
```

```
{'ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there', 'about', 'once', 'during',  
'out', 'very', 'having', 'with', 'they', 'own', 'an', 'be', 'some', 'for', 'do', 'its', 'yours', 'such',  
'into', 'of', 'most', 'itself', 'other', 'off', 'is', 's', 'am', 'or', 'who', 'as', 'fr  
>>> om', 'him', 'each', 'the', 'themselves', 'until', 'below', 'are', 'we', 'these', 'your', 'his',  
'through', 'don', 'nor', 'me', 'were', 'her', 'more', 'himself', 'this', 'down', 'should', 'our',  
'their', 'while', 'above', 'both', 'up', 'to', 'ours', 'had', 'she', 'all', 'no', 'when', 'at', 'any',  
'before', 'them', 'same', 'and', 'been', 'have', 'in', 'will', 'on', 'does', 'yourselves', 'then', 'that',  
'because', 'what', 'over', 'why', 'so', 'can', 'did', 'not', 'now', 'under', 'he', 'you', 'herself',  
'has', 'just', 'where', 'too', 'only', 'myself', 'which', 'those', 'i', 'after', 'few', 'whom', 't',  
'being', 'if', 'theirs', 'my', 'against', 'a', 'by', 'doing', 'it', 'how', 'further', 'was', 'here', 'than'}
```

3. Lemmatizing with NLTK

A very similar operation to stemming is called lemmatizing. The major difference between these is, as you saw earlier, stemming can often create non-existent words, whereas lemmas are actual words.

So, your root stem, meaning the word you end up with, is not something you can just look up in a dictionary, but you can look up a lemma.

Some times you will wind up with a very similar word, but sometimes, you will wind up with a completely different word. Let's see some examples.

```
from nltk.stem import WordNetLemmatizer
```

```
lemmatizer = WordNetLemmatizer()
```

```
print(lemmatizer.lemmatize("cats"))  
print(lemmatizer.lemmatize("cacti"))  
print(lemmatizer.lemmatize("geese"))  
print(lemmatizer.lemmatize("rocks"))  
print(lemmatizer.lemmatize("python"))  
print(lemmatizer.lemmatize("better", pos="a"))  
print(lemmatizer.lemmatize("best", pos="a"))  
print(lemmatizer.lemmatize("run"))  
print(lemmatizer.lemmatize("run", 'v'))
```

Here, we've got a bunch of examples of the lemma for the words that we use. The only major thing to note is that lemmatize takes a part of speech parameter, "pos." If not supplied, the default is "noun." This means that an attempt will be made to find the closest noun, which can create trouble for you. Keep this in mind if you use lemmatizing!

4. Use of Sklearn

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification , regression and clustering algorithms

including support vector machines , random forests, gradient boosting, *k*-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

```
from sklearn.feature_extraction.text import CountVectorizer
```

Convert a collection of text documents to a matrix of token counts. This implementation produces a sparse representation of the counts using `scipy.sparse.csr_matrix`. If you do not provide an a-priori dictionary and you do not use an analyzer that does some kind of feature selection then the number of features will be equal to the vocabulary size found by analyzing the data.

```
from sklearn.metrics.pairwise import euclidean_distances
```

Considering the rows of X (and $Y=X$) as vectors, compute the distance matrix between each pair of vectors.

For efficiency reasons, the euclidean distance between a pair of row vector x and y is computed as:

```
dist(x, y) = sqrt(dot(x, x) - 2 * dot(x, y) + dot(y, y))
```

This formulation has two advantages over other ways of computing distances. First, it is computationally efficient when dealing with sparse data. Second, if one argument varies but the other remains unchanged, then `dot(x, x)` and/or `dot(y, y)` can be pre-computed.

However, this is not the most precise way of doing this computation, and the distance matrix returned by this function may not be exactly symmetric as required by, e.g., `scipy.spatial.distance` functions.

5. Measuring the Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any other angle in the interval $[0, 2\pi)$. It is thus a judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude. Cosine similarity is particularly used in positive space, where the outcome is neatly bounded in $[0, 1]$. The name derives from the term "direction cosine": in this case, note that unit vectors are maximally "similar" if they're parallel and maximally "dissimilar" if they're orthogonal (perpendicular). This is analogous to the cosine, which is unity (maximum value) when the segments subtend a zero angle and zero (uncorrelated) when the segments are perpendicular.

Note that these bounds apply for any number of dimensions, and cosine similarity is most commonly used in high-dimensional positive spaces. For example, in information retrieval and text mining, each term is notionally assigned a different dimension and a document is characterised by a vector where the value of each dimension corresponds to the number of times that term appears in the document. Cosine similarity then gives a useful measure of how similar two documents are likely to be in terms of their subject matter.

The technique is also used to measure cohesion within clusters in the field of Data Mining.

Cosine distance is a term often used for the complement in positive space, that is: $D_c(A, B) = 1 - S_c(A, B)$ where D_c is the cosine distance and S_c is the cosine similarity. It is important to note, however, that this is not a proper distance metric as it does not have the triangle inequality property—or, more formally, the Schwarz inequality—and it violates the coincidence axiom; to repair the triangle inequality property while maintaining the same ordering, it is necessary to convert to angular distance (see below.)

One of the reasons for the popularity of cosine similarity is that it is very efficient to evaluate, especially for sparse vectors, as only the non-zero dimensions need to be considered.

The cosine of two non-zero vectors can be derived by using the Euclidean dot product formula:

$$a \cdot b = \|a\| \|b\| \cos(\theta)$$

Given two vectors of attributes, A and B , the cosine similarity, $\cos(\theta)$, is represented using a dot product and magnitude as

$$\text{Similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n (A_i B_i)}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

where A_i and B_i are components of vector A and B respectively. The resulting similarity ranges from -1 meaning exactly opposite, to 1 meaning exactly the same, with 0 indicating orthogonality (decorrelation), and in-between values indicating intermediate similarity or dissimilarity.

For text matching, the attribute vectors A and B are usually the term frequency vectors of the documents. The cosine similarity can be seen as a method of normalizing document length during comparison.

Cosine Similarity for Vector Space Model

The Dot Product

Let's begin with the definition of the dot product for two vectors: $\vec{a} = (a_1, a_2, a_3, \dots)$ and $\vec{b} = (b_1, b_2, b_3, \dots)$, where a_n and b_n are the components of the vector (features of the document, or TF-IDF values for each word of the document in our example) and the n is the dimension of the vectors:

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$$

As you can see, the definition of the dot product is a simple multiplication of each component from the both vectors added together. See an example of a dot product for two vectors with 2 dimensions each (2D):

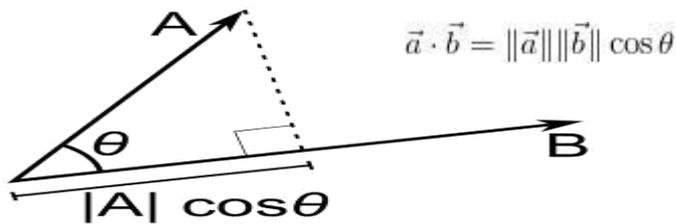
$$\vec{a} = (0, 3)$$

The first thing you probably noticed is that the result of a dot product between two vectors isn't another vector but a single value, a scalar.

$$\vec{b} = (1, 0)$$

$$\vec{a} \cdot \vec{b} = 0 * 1 + 3 * 0 = 0$$

This is all very simple and easy to understand, but what is a dot product ? What is the intuitive idea behind it ? What does it mean to have a dot product of zero ? To understand it, we need to understand what is the geometric definition of the dot product:

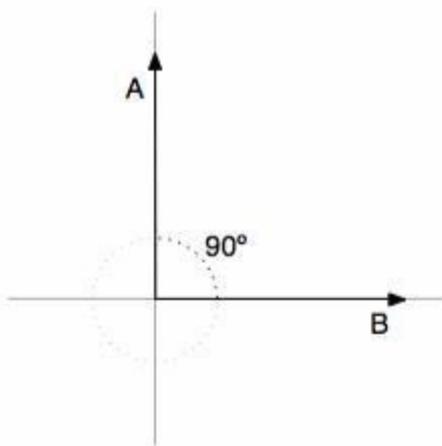


Rearranging the equation to understand it better using the commutative property, we have:

$$\vec{a} \cdot \vec{b} = \|\vec{b}\| \|\vec{a}\| \cos \theta$$

So, what is the term $\|\vec{a}\| \cos \theta$? This term is the projection of the vector \vec{a} into the vector \vec{b} as shown on the image below:

Now, what happens when the vector \vec{a} is orthogonal (with an angle of 90 degrees) to the vector \vec{b} like on the image below ?



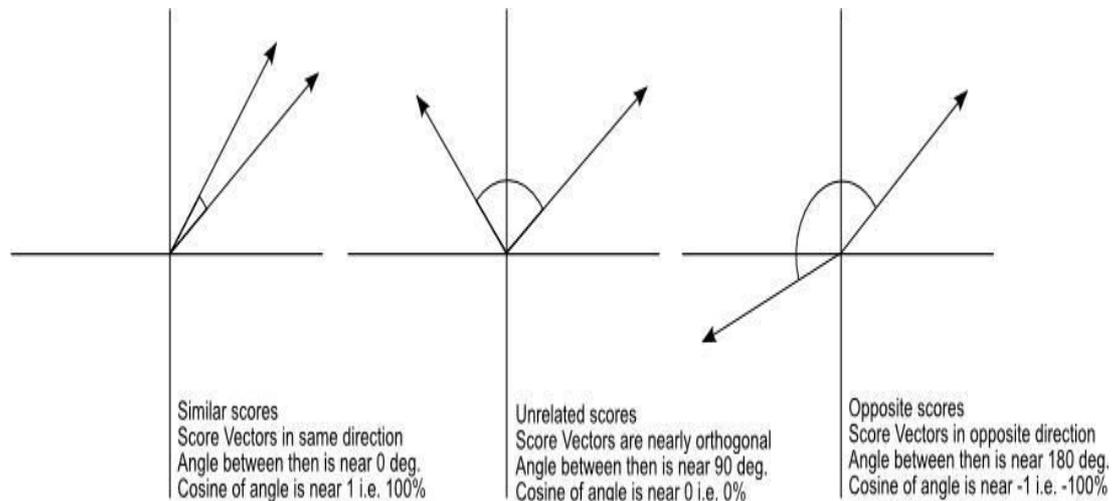
Two orthogonal vectors (with 90 degrees angle).

There will be no adjacent side on the triangle, it will be equivalent to zero, the

term $\|\vec{a}\| \cos \theta$ will be zero and the resulting multiplication with the magnitude of the vector \vec{b} will also be zero. Now you know that, when the dot product between two different vectors is zero, they are orthogonal to each other (they have an angle of 90 degrees), this is a very neat way to check the orthogonality of different vectors. It is also important to note that we are using 2D examples, but the most amazing fact about it is that we can also calculate angles and similarity between vectors in higher dimensional spaces, and that is why math let us see far than the

obvious even when we can't visualize or imagine what is the angle between two vectors with twelve dimensions for instance.

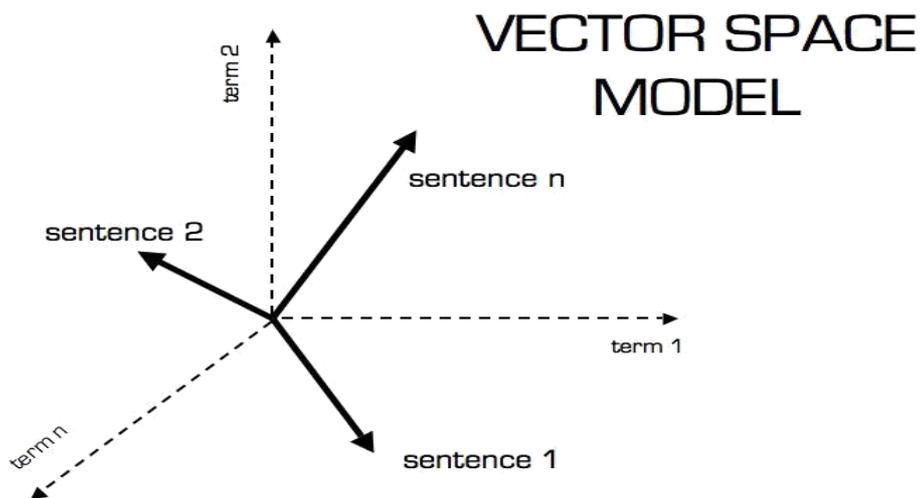
And that is it, this is the cosine similarity formula. Cosine Similarity will generate a metric that says how related are two documents by looking at the angle instead of magnitude, like in the examples below:



The Cosine Similarity values for different documents, 1 (same direction), 0 (90 deg.), -1 (opposite directions).

Note that even if we had a vector pointing to a point far from another vector, they still could have an small angle and that is the central point on the use of Cosine Similarity, the measurement tends to ignore the higher term count on documents. Suppose we have a document with the word "sky" appearing 200 times and another document with the word "sky" appearing 50, the Euclidean distance between them will be higher but the angle will still be small because they are pointing to the same direction, which is what matters when we are comparing documents.

Now that we have a Vector Space Model of documents (like on the image below) modeled as vectors (with TF-IDF counts) and also have a formula to calculate the similarity between different documents in this space, let's see now how we do it in practice using scikit-learn (sklearn).



Vector Space Model

Practice Using Scikit-learn (sklearn)

* In this tutorial I'm using the Python 2.7.5 and Scikit-learn 0.14.1.

The first thing we need to do is to define our set of example documents:

```

1. documents = (
2. "The sky is blue",
3. "The sun is bright",
4. "The sun in the sky is bright",
5. "We can see the shining sun, the bright sun"
6. )

```

And then we instantiate the Sklearn TF-IDF Vectorizer and transform our documents into the TF-IDF matrix:

```

1. from sklearn.feature_extraction.text import TfidfVectorizer
2. tfidf_vectorizer = TfidfVectorizer()
3. tfidf_matrix = tfidf_vectorizer.fit_transform(documents)
4. print tfidf_matrix.shape
5. (4, 11)

```

Now we have the TF-IDF matrix (**tfidf_matrix**) for each document (the number of rows of the matrix) with 11 tf-idf terms (the number of columns from the matrix), we can calculate the Cosine Similarity between the first document (“The sky is blue”) with each of the other documents of the set:

```
1. from sklearn.metrics.pairwise import cosine_similarity
2. cosine_similarity(tfidf_matrix[0:1], tfidf_matrix)
3. array([[ 1. , 0.36651513, 0.52305744, 0.13448867]])
```

The **tfidf_matrix[0:1]** is the Scipy operation to get the first row of the sparse matrix and the resulting array is the Cosine Similarity between the first document with all documents in the set. Note that the first value of the array is 1.0 because it is the Cosine Similarity between the first document with itself. Also note that due to the presence of similar words on the third document (“The sun in the sky is bright”), it achieved a better score.

If you want, you can also solve the Cosine Similarity for the angle between vectors:

$$\cos \theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

We only need to isolate the angle (θ) and move the **cos** to the right hand of the equation:

$$\theta = \arccos \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

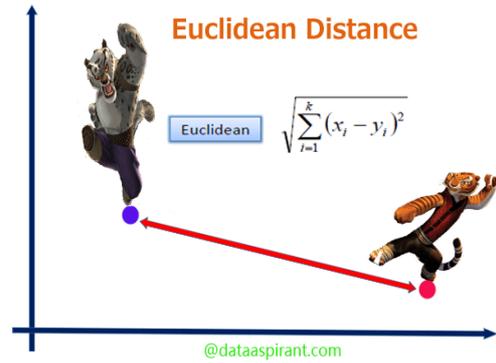
The **arccos** is the same as the inverse of the cosine (\cos^{-1}).

Lets for instance, check the angle between the first and third documents:

```
1. import math
2. # This was already calculated on the previous step, so we just use the value
3. cos_sim = 0.52305744
4. angle_in_radians = math.acos(cos_sim)
5. print math.degrees(angle_in_radians)
6. 58.462437107432784
```

Euclidean Distance

Euclidean distance is the most common use of distance. In most cases when people said about distance, they will refer to Euclidean distance. Euclidean distance is also known as simply distance. When data is dense or continuous, this is the best proximity measure.



The Euclidean distance between two points is the length of the path connecting them. The Pythagorean theorem gives this distance between two points.

The **Euclidean distance** between points \mathbf{p} and \mathbf{q} is the length of the **line segment** connecting them (pq).

In Cartesian coordinates, if $\mathbf{p} = (p_1, p_2, \dots, p_n)$ and $\mathbf{q} = (q_1, q_2, \dots, q_n)$ are two points in Euclidean n -space, then the distance (d) from \mathbf{p} to \mathbf{q} , or from \mathbf{q} to \mathbf{p} is given by the Pythagorean formula:

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$$
$$= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}.$$

DIFFERENCE BETWEEN COSINE SIMILARITY AND EUCLIDEAN DISTANCE SIMILARITY

EUCLIDEAN DISTANCE SIMILARITY

Euclidean Similarity calculates the distance between two users and then it tries to find out the similarity. This makes sense if you think of users as points when there are many dimensions (as many dimensions as the items), whose coordinates are preference values. This similarity metric calculates the Euclidean Distance (d) between two such user points. If you look at User 1, the distance is calculated as 0, because for this particular user the distance is 0. Similarity will be calculated using the formula, $1/1+d$, where d is the distance.

COSINE SIMILARITY

Cosine similarity metric finds the normalized dot product of the two attributes. By determining the cosine similarity, we would effectively try to find the cosine of the angle between the two objects. The cosine of 0° is 1, and it is less than 1 for any other angle.

It is thus a judgement of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude.

Cosine similarity is particularly used in positive space, where the outcome is neatly bounded in $[0,1]$. One of the reasons for the popularity of cosine similarity is that it is very efficient to evaluate, especially for sparse vectors.

Ranking of Answers by the use of Text Classification with NLTK

This psection demonstrates how to obtain an n by n matrix of pairwise semantic/cosine similarity among n text documents. Finding cosine similarity is a basic technique in text mining. The purpose of doing this is to operationalize “common ground” between actors in online political discussion (for more see Liang, 2014, p. 160).

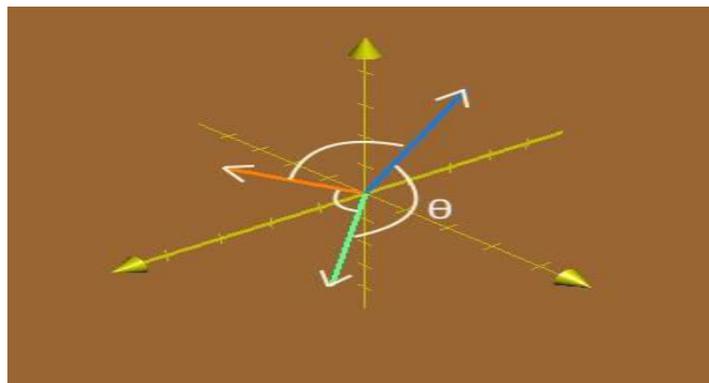
The tools are Python libraries scikit-learn (version 0.18.1; Pedregosa et al., 2011) and nltk (version 3.2.2.; Bird, Klein, & Loper, 2009). Perone’s (2011a; 2011b; 2013) three-piece web tutorial is extremely helpful in explaining the concepts and mathematical logics. However, some of these contents have not kept up with scikit-learn’s recent update and text preprocessing was not included. This post addresses these issues.

If you are familiar with cosine similarity and more interested in the Python part, feel free to skip and scroll down to Section III.

I. What’s going on here?

The cosine similarity is the cosine of the angle between two vectors. Figure 1 shows three 3-dimensional vectors and the angles between each pair. In text analysis, each vector can represent a document. The greater the value of θ , the less the value of $\cos \theta$, thus the less the similarity between two documents.

Figure 1. Three 3-dimensional vectors and the angles between each pair. Blue vector: (1, 2, 3); Green vector: (2, 2, 1); Orange vector: (2, 1, 2).



$$\cos \theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \cdot \|\vec{b}\|}$$

In math equation:

where cosine is the dot/scalar product of two vectors divided by the product of their

Euclidean norms.

II. How to quantify texts in order to do the math?

a. Raw texts are preprocessed with the most common words and punctuation removed, tokenization, and stemming (or lemmatization).

b. A dictionary of unique terms found in the whole corpus is created. Texts are quantified first by calculating the term frequency (tf) for each document. The numbers are used to create a vector for each document where each component in the vector stands for the term frequency in that document. Let n be the number of documents and m be the number of unique terms. Then we have an n by m tf matrix.

c. The core of the rest is to obtain a “term frequency-inverse document frequency” (tf-idf) matrix. Inverse document frequency is an adjustment to term frequency. This

adjustment deals with the problem that generally speaking certain terms do occur more than others. Thus, tf-idf scales up the importance of rarer terms and scales down the importance of more frequent terms relative to the whole corpus.

The idea of the weighting effect of tf-idf is better expressed in the two equations below (the formula for idf is the default one used by scikit-learn (Pedregosa et al., 2011): the 1 added to the denominator prevents division by 0, the 1 added to the nominator makes sure the value of the ratio is greater than or equal to 1, the third 1 added makes sure that idf is greater than 0, i.e., for an extremely common term t for which $n = df(d,t)$, its idf is at least not 0 so that its tf still matters; Note that in Perone (2011b) there is only one 1 added to the denominator, which results in negative values after taking the logarithm for some cases. Negative value is difficult to interpret):

$$\text{tf-idf} = \text{tf} \times \text{idf} \quad (1)$$

$$\text{idf}(t) = \log \frac{n+1}{df(d,t)+1} + 1 \quad (2)$$

where n is the total number of documents and $df(d, t)$ is **the number of documents in which term t appears**. In Equation 2, as $df(d, t)$ gets smaller, $\text{idf}(t)$ gets larger. In Equation 1, tf is a local parameter for individual documents, whereas idf is a global parameter taking the whole corpus into account.

Therefore, even the tf for one term is very high for document d_1 , if it appears frequently in other documents (with a smaller idf), its importance of “defining” d_1 is scaled down. On the other hand, if a term has high tf in d_1 and does not appear in other documents (with a greater idf), it becomes an important feature that distinguishes d_1 from other documents.

d. The calculated tf-idf is normalized by the Euclidean norm so that each row vector has a length of 1. The normalized tf-idf matrix should be in the shape of n by m . A cosine similarity matrix (n by n) can be obtained by multiplying the if-idf matrix by its transpose (m by n).

Implementation of Problem

A simple real-world data for the implementation is obtained from the movie review corpus provided by nltk (Pang & Lee, 2004). The first two reviews from the positive set and the negative set are selected. Then the first sentence of these for reviews are selected. We can first define 4 documents in Python as:

```
d1 = "plot: two teen couples go to a church party, drink and then drive."  
d2 = "films adapted from comic books have had plenty of success , whether they're about  
1 superheroes ( batman , superman , spawn ) , or geared toward kids ( casper ) or the arthouse  
2 crowd ( ghost world ) , but there's never really been a comic book like from hell before . "  
3 d3 = "every now and then a movie comes along from a suspect studio , with every indication  
4 that it will be a stinker , and to everybody's surprise ( perhaps even the studio ) the film  
5 becomes a critical darling . "  
d4 = "damn that y2k bug . "  
documents = [d1, d2, d3, d4]
```

a. Preprocessing with nltk

The default functions of CountVectorizer and TfidfVectorizer in scikit-learn detect word boundary and remove punctuations automatically. However, if we want to do stemming or lemmatization, we need to customize certain parameters in CountVectorizer and TfidfVectorizer. **Doing this overrides the default tokenization setting, which means that we have to customize tokenization, punctuation removal, and turning terms to lower case altogether.**

Normalize by stemming:

```
import nltk, string, numpy  
1 nltk.download('punkt') # first-time use only  
2 stemmer = nltk.stem.porter.PorterStemmer()  
3 def StemTokens(tokens):  
4     return [stemmer.stem(token) for token in tokens]  
5 remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)  
6 def StemNormalize(text):  
7     return StemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))  
8
```

Normalize by lemmatization:

```
nltk.download('wordnet') # first-time use only  
1 lemmer = nltk.stem.WordNetLemmatizer()  
2 def LemTokens(tokens):  
3     return [lemmer.lemmatize(token) for token in tokens]  
4 remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)  
5 def LemNormalize(text):  
6     return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))  
7
```

If we want more meaningful terms in their dictionary forms, lemmatization is preferred.

b. Turn text into vectors of term frequency:

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 LemVectorizer = CountVectorizer(tokenizer=LemNormalize, stop_words='english')
3 LemVectorizer.fit_transform(documents)
```

Normalized (after lemmatization) text in the four documents are tokenized and each term is indexed:

```
print LemVectorizer.vocabulary_
```

Out:

```
{u'spawn': 29, u'crowd': 11, u'casper': 5, u'church': 6, u'hell': 20,
u'comic': 8, u'superheroes': 33, u'superman': 34, u'plot': 27, u'movie': 24,
u'book': 3, u'suspect': 36, u'film': 17, u'party': 25, u'darling': 13, u'really': 28,
u'teen': 37, u'everybodys': 16, u'damn': 12, u'batman': 2, u'couple': 9, u'drink': 14,
u'like': 23, u'geared': 18, u'studio': 31, u'plenty': 26, u'surprise': 35, u'world': 39,
u'come': 7, u'bug': 4, u'kid': 22, u'ghost': 19, u'arthouse': 1, u'y2k': 40,
u'stinker': 30, u'success': 32, u'drive': 15, u'theyre': 38, u'indication': 21,
u'critical': 10, u'adapted': 0}
```

And we have the tf matrix:

```
tf_matrix = LemVectorizer.transform(documents).toarray()
```

```
print tf_matrix
```

Out:

```
[[0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
 1 0 0 0]
 [1 1 1 2 0 1 0 0 2 0 0 1 0 0 0 0 0 1 1 1 1 0 1 1 0 0 1 0 1 1 0 0 1 1 1 0 0
 0 1 1 0]
 [0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0 0 0 1 2 0 0 0 1 1
 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1]]
```

This should be a 4 (# of documents) by 41 (# of terms in the corpus). Check its shape:

```
tf_matrix.shape
```

Out:

```
(4, 41)
```

c. Calculate idf and turn tf matrix to tf-idf matrix:

Get idf:

```
1 from sklearn.feature_extraction.text import TfidfTransformer
2 tfidfTran = TfidfTransformer(norm="l2")
3 tfidfTran.fit(tf_matrix)
4 print tfidfTran.idf_
```

Out:

```
1 [ 1.91629073  1.91629073  1.91629073  1.91629073  1.91629073  1.91629073
2  1.91629073  1.91629073  1.91629073  1.91629073  1.91629073  1.91629073
3  1.91629073  1.91629073  1.91629073  1.91629073  1.91629073  1.51082562
4  1.91629073  1.91629073  1.91629073  1.91629073  1.91629073  1.91629073
5  1.91629073  1.91629073  1.91629073  1.91629073  1.91629073  1.91629073
6  1.91629073  1.91629073  1.91629073  1.91629073  1.91629073  1.91629073
7  1.91629073  1.91629073  1.91629073  1.91629073  1.91629073]
```

Now we have a vector where each component is the idf for each term. In this case, the values are almost the same because other than one term, each term only appears in 1 document. The exception is the 18th term that appears in 2 document.

d. Get the tf-idf matrix (4 by 41):

```
tfidf_matrix = tfidfTran.transform(tf_matrix)
print tfidf_matrix.toarray()
```

Here what the transform method does is multiplying the tf matrix (4 by 41) by the diagonal idf matrix (41 by 41 with idf for each term on the main diagonal), and dividing the tf-idf by the Euclidean norm. This output takes too much space and you can check it by yourself.

e. Get the pairwise similarity matrix (n by n):

```
cos_similarity_matrix = (tfidf_matrix * tfidf_matrix.T).toarray()
print cos_similarity_matrix
```

Out:

```
array([[ 1.      ,  0.      ,  0.      ,  0.      ],
       [ 0.      ,  1.      ,  0.03264186,  0.      ],
       [ 0.      ,  0.03264186,  1.      ,  0.      ],
       [ 0.      ,  0.      ,  0.      ,  1.      ]])
```

The matrix obtained in the last step is multiplied by its transpose. The result is the similarity matrix, which indicates that d2 and d3 are more similar to each other than any other pair.

Sample Output

Sample 1:

Step 1: This is user interface generated after running the code:

My Application

Question-Answer System

Question:

Answer 1:

Answer 2:

Answer 3:

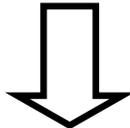
Answer 4:

Answer 5:

INSERT

EUCLIDEAN SIMILARITY MEASURE

COSINE SIMILARITY MEASURE



Step 2: After clicking the Insert button

My Application

Question-Answer System

Question: How should I start learning data science from ground level? Is it worth reading the book Data Science from Scratch? For learning data science, is Data Camp better than Coursera's data science specialization?

Answer 1: As is true for learning any skill, though, you want to start practicing as soon as possible. Download either R or Python and start doing drills with either language. Personally, I'd recommend Python — it's a more widely used language and is versatile, so if you want to pivot and start building

Answer 2: Data Science everyone is talking about it in town. Because of its exciting career is attracting every one toward it. So like you everyone is searching for the best information available about Data Science on the internet. At this stage you might have got some information about Data Science

Answer 3: I think it depends on what you want to do with Data Science and your current level. Although you say 'ground level', everybody has their own experience. As other answers explain very well, Data Science is a very broad topic that contains several subfields such as Statistics, Machine Learning

Answer 4: I suggest you to also start working on the data. Reading can help you learn the concepts but working on real data can help with you to understand the concepts better. You could check Analytica TreasureHunt(R), which has multiple courses listed ranging from basic fundamental concepts

Answer 5: I wouldn't focus so much on learning statistics "for data science", but more on just "learning statistics". Data Science itself is a combination of two fields, statistics/mathematics and computer science. There were "data scientists" that sat at the intersection of those two fields far before

INSERT

EUCLIDEAN SIMILARITY MEASURE

COSINE SIMILARITY MEASURE

Step 3: Now, to measure cosine similarity between the question and the answers we have to click the “Cosine Similarity Measure” button to get the desired output

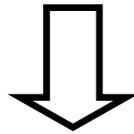
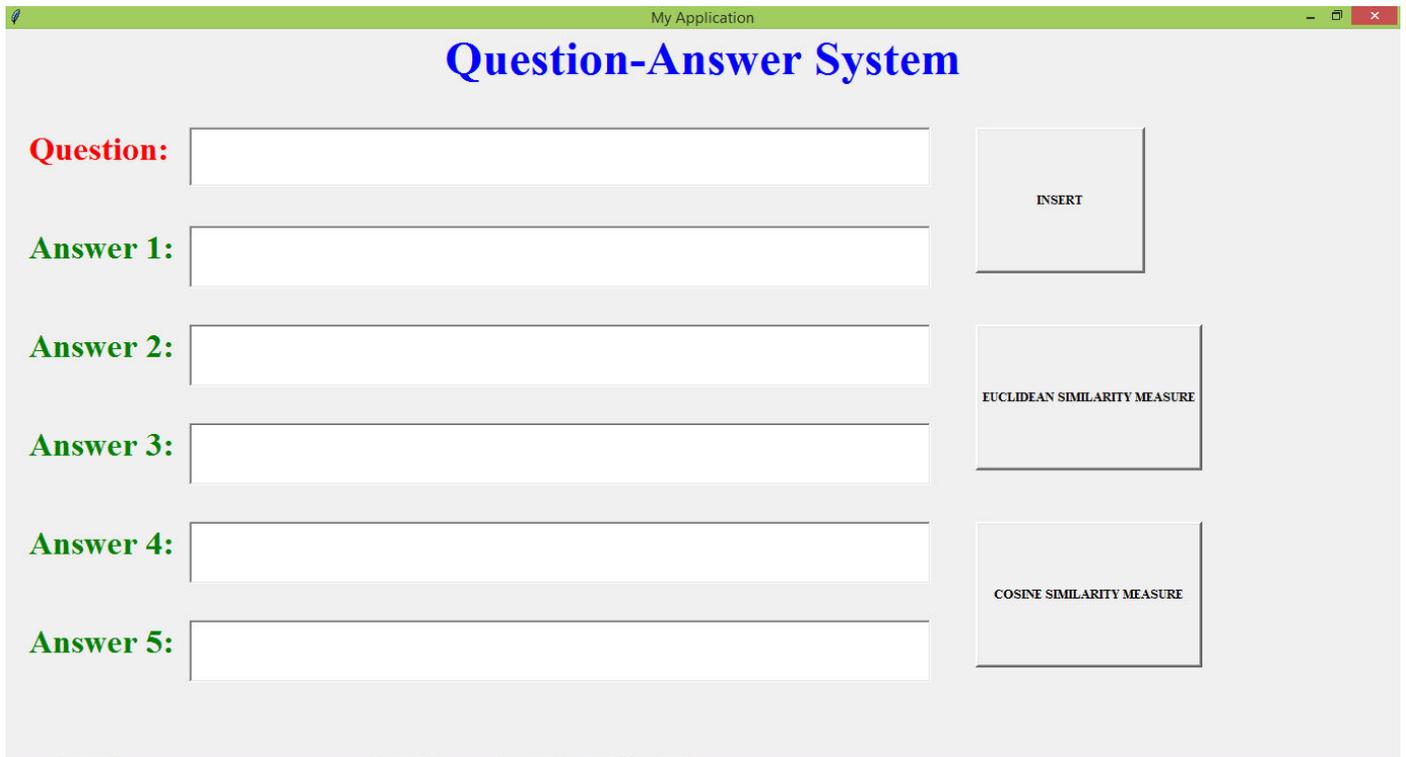
```
Windows Powershell
PS C:\Users\rohit54\Desktop\nlp\New folder> python search.py
{'start': 99, 'learning': 60, 'data': 32, 'science': 87, 'ground': 49, 'level': 62, 'worth': 123, 'reading': 79, 'book':
13, 'scratchfor': 89, 'camp': 16, 'better': 12, 'courseras': 28, 'specialization': 97, 'true': 113, 'skill': 95, 'want':
118, 'practicing': 73, 'soon': 96, 'possible': 72, 'download': 35, 'r': 77, 'python': 75, 'doing': 34, 'drill': 36, 'l
anguage': 58, 'personally': 70, '': 127, 'd': 31, 'recommend': 81, '-': 125, 's': 84, 'widely': 120, 'used': 115, 'vers
atile': 116, 'pivot': 71, 'building': 15, 'web': 119, 'apps': 7, 'll': 65, 'shorter': 93, 'curve': 30, 'talking': 106, '
town': 111, 'exciting': 38, 'career': 17, 'attracting': 8, 'like': 63, 'searching': 90, 'best': 11, 'information': 54, '
available': 9, 'internet': 55, 'stage': 98, 'got': 48, 'referred': 82, 'friend': 45, 'colleague': 20, 'senior': 91, 'hav
ing': 51, 'question': 76, 'mind': 67, 'regarding': 83, 'prerequisite': 74, 'shift': 92, 'new': 69, 'technology': 107, 't
hink': 109, 'depends': 33, 'current': 29, 'say': 86, '': 126, 'everybody': 37, 'ha': 50, 'experience': 39, 'answer': 4,
'explain': 40, 'broad': 14, 'topic': 110, 'contains': 25, 'subfields': 103, 'statistic': 100, 'machine': 66, 'computer':
22, 'good': 47, 'approach': 6, 'subfield': 102, 'feel': 42, 'comfortable': 24, 'inclined': 53, 'learn': 59, 'suggest':
104, 'working': 122, 'help': 52, 'concept': 23, 'real': 80, 'understand': 114, 'check': 18, 'analyttica': 3, 'treasureh
untr': 112, 'multiple': 68, 'course': 27, 'listed': 64, 'ranging': 78, 'basic': 10, 'fundamental': 46, 'advanced': 0, 'a
nalytics': 2, 'simulation': 94, 'apart': 5, 'content': 26, 'allows': 1, 'work': 121, 'learnt': 61, 'wouldn': 124, 't': 1
05, 'focus': 44, '": 128, '": 129, 'just': 57, 'combination': 21, 'field': 43, 'statisticsmathematics': 101, 'scientis
t': 88, 'sat': 85, 'intersection': 56, 'far': 41, 'term': 108, 'wa': 117, 'coined': 19}
[[ 1.          0.07843046  0.3239289  0.37201804  0.13317064  0.29719716]
 [ 0.07843046  1.          0.          0.10812055  0.02029532  0.06371181]
 [ 0.3239289   0.          1.          0.14074849  0.03774485  0.15437385]
 [ 0.37201804  0.10812055  0.14074849  1.          0.05476291  0.18445805]
 [ 0.13317064  0.02029532  0.03774485  0.05476291  1.          0.02953485]
 [ 0.29719716  0.06371181  0.15437385  0.18445805  0.02953485  1.          ]]
PS C:\Users\rohit54\Desktop\nlp\New folder>
```

Step 4: To measure the Euclidean similarity between the question and the answers we have to click “Euclidean Similarity Measure” button to get the desired output

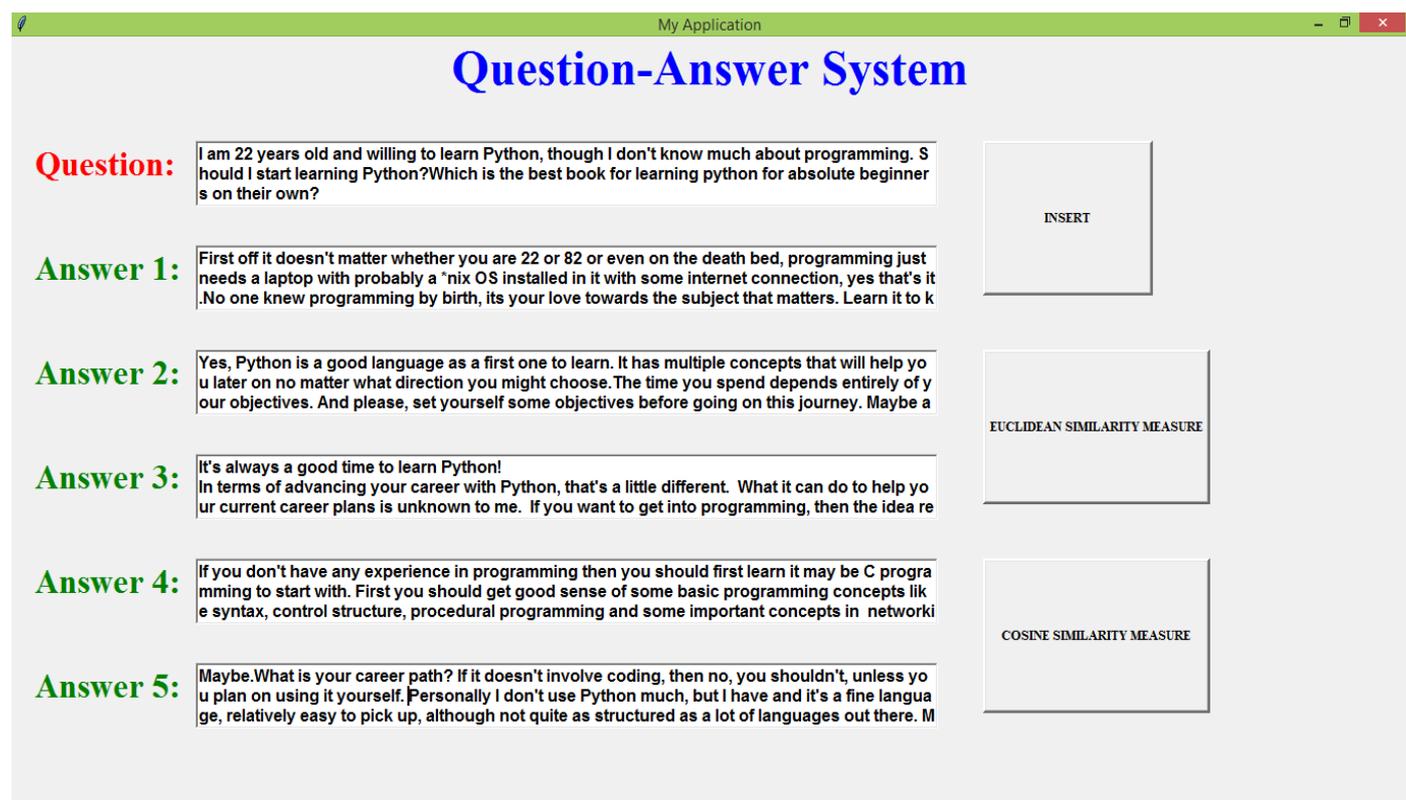
```
Windows Powershell
PS C:\Users\rohit54\Desktop\nlp\New folder> python test.py
{'start': 99, 'learning': 60, 'data': 32, 'science': 87, 'ground': 49, 'level': 62, 'worth': 123, 'reading': 79, 'book':
13, 'scratchfor': 89, 'camp': 16, 'better': 12, 'courseras': 28, 'specialization': 97, 'true': 113, 'skill': 95, 'want':
118, 'practicing': 73, 'soon': 96, 'possible': 72, 'download': 35, 'r': 77, 'python': 75, 'doing': 34, 'drill': 36, 'l
anguage': 58, 'personally': 70, '': 127, 'd': 31, 'recommend': 81, '-': 125, 's': 84, 'widely': 120, 'used': 115, 'vers
atile': 116, 'pivot': 71, 'building': 15, 'web': 119, 'apps': 7, 'll': 65, 'shorter': 93, 'curve': 30, 'talking': 106, '
town': 111, 'exciting': 38, 'career': 17, 'attracting': 8, 'like': 63, 'searching': 90, 'best': 11, 'information': 54, '
available': 9, 'internet': 55, 'stage': 98, 'got': 48, 'referred': 82, 'friend': 45, 'colleague': 20, 'senior': 91, 'hav
ing': 51, 'question': 76, 'mind': 67, 'regarding': 83, 'prerequisite': 74, 'shift': 92, 'new': 69, 'technology': 107, 't
hink': 109, 'depends': 33, 'current': 29, 'say': 86, '': 126, 'everybody': 37, 'ha': 50, 'experience': 39, 'answer': 4,
'explain': 40, 'broad': 14, 'topic': 110, 'contains': 25, 'subfields': 103, 'statistic': 100, 'machine': 66, 'computer':
22, 'good': 47, 'approach': 6, 'subfield': 102, 'feel': 42, 'comfortable': 24, 'inclined': 53, 'learn': 59, 'suggest':
104, 'working': 122, 'help': 52, 'concept': 23, 'real': 80, 'understand': 114, 'check': 18, 'analyttica': 3, 'treasureh
untr': 112, 'multiple': 68, 'course': 27, 'listed': 64, 'ranging': 78, 'basic': 10, 'fundamental': 46, 'advanced': 0, 'a
nalytics': 2, 'simulation': 94, 'apart': 5, 'content': 26, 'allows': 1, 'work': 121, 'learnt': 61, 'wouldn': 124, 't': 1
05, 'focus': 44, '": 128, '": 129, 'just': 57, 'combination': 21, 'field': 43, 'statisticsmathematics': 101, 'scientis
t': 88, 'sat': 85, 'intersection': 56, 'far': 41, 'term': 108, 'wa': 117, 'coined': 19}
[[ 0.]]
[[ 10.]]
[[ 7.48331477]]
[[ 7.28010989]]
[[ 9.79795897]]
[[ 7.54983444]]
PS C:\Users\rohit54\Desktop\nlp\New folder>
```

Sample 2:

Step 1: This is user interface generated after running the code:



Step 2: After clicking the Insert button



Step 3: Now, to measure cosine similarity between the question and the answers we have to click the “Cosine Similarity Measure” button to get the desired output

```
Windows Powershell
PS C:\Users\rohit54\Desktop\nlp\New folder> python search.py
{'22': 0, 'year': 184, 'old': 111, 'willing': 179, 'learn': 87, 'python': 126, 'dont': 41, 'know': 83, 'programming': 125, 'start': 150, 'learning': 88, 'pythonwhich': 129, 'best': 11, 'book': 14, 'absolute': 2, 'beginner': 9, 'doesnt': 40, 'matter': 96, '82': 1, 'death': 29, 'bed': 8, 'just': 80, 'need': 104, 'laptop': 85, 'probably': 121, 'nix': 108, 'o': 109, 'installed': 70, 'internet': 73, 'connection': 22, 'yes': 185, 'thats': 162, 'itno': 76, 'knew': 82, 'birth': 13, 'love': 93, 'subject': 155, 'better': 12, 'tick': 164, 'box': 15, 'checklist': 18, 'think': 163, 'c': 16, 'course': 25, 'make': 95, 'programmer': 124, 'feel': 53, 'wrong': 183, 'look': 91, 'httpswwkagglecom': 65, 'machine': 94, 'playground': 120, 'physicist': 117, 'electrical': 45, 'engineer': 46, 'developer': 34, 'awesome': 6, 'software': 147, 'like': 89, 'scikitlearn': 141, 'sympy': 158, 'astronomer': 5, 'mechanical': 100, 'suggest': 156, 'good': 57, 'language': 84, 'ha': 58, 'multiple': 102, 'concept': 21, 'help': 62, 'later': 86, 'direction': 37, 'choosethe': 19, 'time': 165, 'spend': 148, 'depends': 31, 'entirely': 47, 'objective': 110, 'set': 144, 'going': 56, 'journey': 78, 'maybe': 97, 'ask': 4, 'write': 182, 'quora': 131, 'stackoverflow': 149, 'network': 105, 'experienced': 51, 'people': 115, 'review': 137, 'sure': 157, 'resonate': 136, 'having': 60, 'stay': 152, 'motivated': 101, 'exactly': 48, 'headed': 61, 'pythonin': 128, 'term': 160, 'advancing': 3, 'career': 17, 'little': 90, 'different': 36, 'current': 27, 'plan': 119, 'unknown': 170, 'want': 177, 'idea': 66, 'really': 134, 'isnt': 75, 'program': 123, 'tool': 166, 'vastness': 176, 'determine': 33, 'getting': 55, 'in stance': 71, 'route': 138, 'web': 178, 'django': 38, 'javascript': 77, 'jquery': 79, 'html': 64, 'cs': 26, 'necessary': 103, 'currently': 28, 'believe': 10, 'ruby': 139, 'rail': 132, 'hot': 63, 'trending': 167, 'development': 35, 'deserves': 32, 'consideration': 23, 'v': 175, 'realize': 133, 'key': 81, 'unlock': 172, 'door': 42, 'new': 107, 'world': 180, 'me an': 99, 'lot': 92, 'effort': 44, 'definitely': 30, 'worth': 181, 'experience': 50, 'sense': 143, 'basic': 7, 'syntax': 159, 'control': 24, 'structure': 153, 'procedural': 122, 'important': 68, 'networking': 106, 'operating': 113, 'scriptin g': 142, 'pythonif': 127, 'familiar': 52, 'install': 69, 'idle': 67, 'online': 112, 'documentation': 39, 'coding': 20, 'starting': 151, 'hard': 59, 'example': 49, 'understanding': 169, 'interactive': 72, 'tutorial': 168, 'site': 146, 'test': 161, 'maybewhat': 98, 'path': 114, 'involve': 74, 'shouldnt': 145, 'unless': 171, 'using': 174, 'personally': 116, 'us e': 173, 'fine': 54, 'relatively': 135, 'easy': 43, 'pick': 118, 'quite': 130, 'structured': 154, 'run': 140}
[[ 1. 0.16391149 0.04667802 0.16861165 0.2260286 0.05173178]
 [ 0.16391149 1. 0.06676364 0.12604656 0.11715771 0.02305848]
 [ 0.04667802 0.06676364 1. 0.10107649 0.08420843 0.05838684]
 [ 0.16861165 0.12604656 0.10107649 1. 0.14573035 0.13238098]
 [ 0.2260286 0.11715771 0.08420843 0.14573035 1. 0.07288313]
 [ 0.05173178 0.02305848 0.05838684 0.13238098 0.07288313 1.]]
PS C:\Users\rohit54\Desktop\nlp\New folder>
```

Step 4: To measure the Euclidean similarity between the question and the answers we have to click “Euclidean Similarity Measure” button to get the desired output

```
Windows Powershell
PS C:\Users\rohit54\Desktop\nlp\New folder> python test.py
{'22': 0, 'year': 184, 'old': 111, 'willing': 179, 'learn': 87, 'python': 126, 'dont': 41, 'know': 83, 'programming': 125, 'start': 150, 'learning': 88, 'pythonwhich': 129, 'best': 11, 'book': 14, 'absolute': 2, 'beginner': 9, 'doesnt': 40, 'matter': 96, '82': 1, 'death': 29, 'bed': 8, 'just': 80, 'need': 104, 'laptop': 85, 'probably': 121, 'nix': 108, 'o': 109, 'installed': 70, 'internet': 73, 'connection': 22, 'yes': 185, 'thats': 162, 'itno': 76, 'knew': 82, 'birth': 13, 'love': 93, 'subject': 155, 'better': 12, 'tick': 164, 'box': 15, 'checklist': 18, 'think': 163, 'c': 16, 'course': 25, 'make': 95, 'programmer': 124, 'feel': 53, 'wrong': 183, 'look': 91, 'httpswwkagglecom': 65, 'machine': 94, 'playground': 120, 'physicist': 117, 'electrical': 45, 'engineer': 46, 'developer': 34, 'awesome': 6, 'software': 147, 'like': 89, 'scikitlearn': 141, 'sympy': 158, 'astronomer': 5, 'mechanical': 100, 'suggest': 156, 'good': 57, 'language': 84, 'ha': 58, 'multiple': 102, 'concept': 21, 'help': 62, 'later': 86, 'direction': 37, 'choosethe': 19, 'time': 165, 'spend': 148, 'depends': 31, 'entirely': 47, 'objective': 110, 'set': 144, 'going': 56, 'journey': 78, 'maybe': 97, 'ask': 4, 'write': 182, 'quora': 131, 'stackoverflow': 149, 'network': 105, 'experienced': 51, 'people': 115, 'review': 137, 'sure': 157, 'resonate': 136, 'having': 60, 'stay': 152, 'motivated': 101, 'exactly': 48, 'headed': 61, 'pythonin': 128, 'term': 160, 'advancing': 3, 'career': 17, 'little': 90, 'different': 36, 'current': 27, 'plan': 119, 'unknown': 170, 'want': 177, 'idea': 66, 'really': 134, 'isnt': 75, 'program': 123, 'tool': 166, 'vastness': 176, 'determine': 33, 'getting': 55, 'in stance': 71, 'route': 138, 'web': 178, 'django': 38, 'javascript': 77, 'jquery': 79, 'html': 64, 'cs': 26, 'necessary': 103, 'currently': 28, 'believe': 10, 'ruby': 139, 'rail': 132, 'hot': 63, 'trending': 167, 'development': 35, 'deserves': 32, 'consideration': 23, 'v': 175, 'realize': 133, 'key': 81, 'unlock': 172, 'door': 42, 'new': 107, 'world': 180, 'me an': 99, 'lot': 92, 'effort': 44, 'definitely': 30, 'worth': 181, 'experience': 50, 'sense': 143, 'basic': 7, 'syntax': 159, 'control': 24, 'structure': 153, 'procedural': 122, 'important': 68, 'networking': 106, 'operating': 113, 'scriptin g': 142, 'pythonif': 127, 'familiar': 52, 'install': 69, 'idle': 67, 'online': 112, 'documentation': 39, 'coding': 20, 'starting': 151, 'hard': 59, 'example': 49, 'understanding': 169, 'interactive': 72, 'tutorial': 168, 'site': 146, 'test': 161, 'maybewhat': 98, 'path': 114, 'involve': 74, 'shouldnt': 145, 'unless': 171, 'using': 174, 'personally': 116, 'us e': 173, 'fine': 54, 'relatively': 135, 'easy': 43, 'pick': 118, 'quite': 130, 'structured': 154, 'run': 140}
[[ 0.]]
[[ 8.60232527]]
[[ 8.36660027]]
[[ 10.34408043]]
[[ 9.38083152]]
[[ 6.92820323]]
PS C:\Users\rohit54\Desktop\nlp\New folder>
```

Conclusion

This document presented a method for measuring the similarity between sentences or very short texts, based on semantic and word order information. Firstly, semantic similarity is derived from a lexical knowledge base and a corpus. The lexical knowledge base models common human knowledge about words in a natural language, this knowledge is usually stable across a wide range of language application areas. A corpus reflects the actual usage of language and words. Thus our semantic similarity not only captures common human knowledge, but it is also able to adapt to an application area using a corpus specific to that application. Secondly, the proposed method considers the impact of word order on sentence meaning. The derived word order similarity measures the number of different words as well as the number of word pairs in a different order. The overall sentence similarity is then defined as a combination of semantic similarity and word order similarity. Considering the view that word order plays a subordinate role for interpreting sentence meaning, we weight word order similarity less in defining the overall sentence similarity. To evaluate our similarity algorithm, we collected a set of 32 sentence pairs from a variety of articles and books in computational linguistics. An initial experiment on this data illustrates that the proposed method provides similarity measures that are fairly consistent with human knowledge. Next we constructed a data set of 30 sentence pairs using a dictionary definition for each of the Rubenstein and Goodenough word pairs. The sentences were rated by human participants as a benchmark for comparison with our method which performed well on this data set. Further work will include the construction of a more varied sentence pair dataset with human ratings and an improvement to the algorithm to disambiguate word sense using the surrounding words to give a little contextual information. Currently comparison with some of the other algorithms discussed is very difficult due to a lack of any other published results on sentence similarities (a benchmark data set) and a variety of problems in re-implementing these algorithms for this domain. These include the substantial amount of parameters which must be manually set and the definition of features.

References

- [I] J. Allen, *Natural Language Understanding*. Benjamin Cummings, Redwood City, CA, 1995.
- [II] <https://sites.temple.edu/tudsc/2017/03/30/measuring-similarity-between-texts-in-python/>
- [III] http://scikit-learn.org/stable/modules/feature_extraction.html#the-bag-of-words-representation
- [IV] A. Budanitsky and G. Hirst, "Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures," *Workshop WordNet and Other Lexical Resources, Second Meeting of the North American Chapter of the Association for Computational Linguistics*, Pittsburgh, 2001.
- [IV] C. Burgess, K. Livesay, and K. Lund, "Explorations in context space: Words, sentences, discourse," *Discourse Processes*, vol. 25, no. 2-3, pp. 211-257, 1998.
- [V] W.G. Charles, "Contextual correlates of meaning," *Applied Psycholinguistics*, vol. 21, no. 4, pp. 505-524, 2000.
- [VI] J. H. Chiang and H.C. Yu, "Literature extraction of protein functions using sentence pattern mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 8, pp. 1088-1098, 2005.
- [VII] T.A.S. Coelho, P.P. Calado, L.V. Souza, B. Ribeiro-Neto, and R. Muntz, "Image retrieval using multiple evidence ranking," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 4, pp. 408-417, 2004.
- [IX] G. Erkan and D.R. Radev, "LexRank: Graph-based lexical centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, vol. 22, pp. 457-477, 2005.
- [X] P.W. Foltz, W. Kintsch, and T.K. Landauer, "The measurement of textual coherence with latent semantic analysis," *Discourse Processes*, vol. 25, no. 2-3, pp. 285-307, 1998.
- [XI] P. Wiemer-Hastings, "Adding syntactic information to LSA," *Proceedings of the Twenty-second Annual Conference of the Cognitive Science Society*, Lawrence Erlbaum Associates, Mahwah, NJ, pp. 989-993, 2000.
- [XII] V. Hatzivassiloglou, J. Klavans, and E. Eskin, "Detecting text similarity over short passages: Exploring linguistic feature combinations via machine learning," *Joint SIGDAT Conference on Empirical Methods in NLP and Very Large Corpora*, University of Maryland, College Park, MD, USA, 1999.
- [XIII] M.A. Rodriguez and M.J. Egenhofer, "Determining semantic similarity among entity classes from different ontologies," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 2, pp. 442-456, 2003.
- [XIV] H. Rubenstein and J.B. Goodenough, "Contextual Correlates of Synonymy," *Comm. ACM*, vol.8, No. 10, pp627-633, 1965.
- [XV] G. Salton, *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Mass.Wokingham, 1989.

APPENDIX

The code snippet for measuring the similarity between question and answers are given here:

Cosine Similarity Measure:

```
File Edit Selection Find View Goto Tools Project Preferences Help
search.py x test.py x grade.py project1.py x abc.py x
1 import nltk
2 from nltk.tokenize import word_tokenize
3 from nltk.stem import WordNetLemmatizer
4 import string
5 from nltk.corpus import stopwords
6 from sklearn.feature_extraction.text import CountVectorizer
7 from sklearn.metrics.pairwise import euclidean_distances
8
9 q = "How should I start learning data science from ground level? Is it worth reading the book Data Science from Scratch?For learning data science, is Data Camp bett
10 a1 = "As is true for learning any skill, though, you want to start practicing as soon as possible. Download either R or Python and start doing drills with either la
11 a2 = "Data Science everyone is talking about it in town. Because of its exciting career is attracting every one toward it. So like you everyone is searching for the
12 a3 = "I think it depends on what you want to do with Data Science and your current level. Although you say 'ground level', everybody has their own experience. As ot
13 a4 = "I suggest you to also start working on the data. Reading can help you learn the concepts but working on real data can help with you to understand the concepts
14 a5 = "I wouldn't focus so much on learning statistics "for data science", but more on just "learning statistics". Data Science itself is a combination of two fields
15 documents = [q, a1, a2, a3, a4, a5]
16
17 lemmer = nltk.stem.WordNetLemmatizer()
18
19 def LemTokens(tokens):
20     return [lemmer.lemmatize(token) for token in tokens]
21
22 remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
23
24 def LemNormalize(text):
25     return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
26
27 LemVectorizer = CountVectorizer(tokenizer=LemNormalize, stop_words='english')
28 features = LemVectorizer.fit_transform(documents).todense()
29 print(LemVectorizer.vocabulary_)
30
31 tf_matrix = LemVectorizer.transform(documents).todense()
32 #print (tf_matrix)
33
34 #print (tf_matrix.shape)
35
36 from sklearn.feature_extraction.text import TfidfTransformer
37 tfidfTran = TfidfTransformer(norm="l2")
38 tfidfTran.fit(tf_matrix)
39 #print (tfidfTran.idf_)
40
41 tfidf_matrix = tfidfTran.transform(tf_matrix)
42 #print (tfidf_matrix.toarray())
43
44 cos_similarity_matrix = (tfidf_matrix * tfidf_matrix.T).todense()
45 print (cos_similarity_matrix)
Line 20, Column 58 Tab Size: 4 Python
```

Euclidean Similarity Measure

```
File Edit Selection Find View Goto Tools Project Preferences Help
search.py x test.py x grade.py project1.py x abc.py x
1 import nltk
2 from nltk.tokenize import word_tokenize
3 from nltk.stem import WordNetLemmatizer
4 import string
5 from nltk.corpus import stopwords
6 from sklearn.feature_extraction.text import CountVectorizer
7 from sklearn.metrics.pairwise import euclidean_distances
8
9
10 q = "How should I start learning data science from ground level? Is it worth reading the book Data Science from Scratch?For learning data science, is Data Camp bett
11 a1 = "As is true for learning any skill, though, you want to start practicing as soon as possible. Download either R or Python and start doing drills with either la
12 a2 = "Data Science everyone is talking about it in town. Because of its exciting career is attracting every one toward it. So like you everyone is searching for the
13 a3 = "I think it depends on what you want to do with Data Science and your current level. Although you say 'ground level', everybody has their own experience. As ot
14 a4 = "I suggest you to also start working on the data. Reading can help you learn the concepts but working on real data can help with you to understand the concepts
15 a5 = "I wouldn't focus so much on learning statistics "for data science", but more on just "learning statistics". Data Science itself is a combination of two fields
16 documents = [q, a1, a2, a3, a4, a5]
17
18 lemmer = nltk.stem.WordNetLemmatizer()
19
20 def LemTokens(tokens):
21     return [lemmer.lemmatize(token) for token in tokens]
22
23 remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
24
25 def LemNormalize(text):
26     return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
27
28 LemVectorizer = CountVectorizer(tokenizer=LemNormalize, stop_words='english')
29 features = LemVectorizer.fit_transform(documents).todense()
30 print(LemVectorizer.vocabulary_)
31
32
33 for f in features:
34     a = ( euclidean_distances(features[0], f)
35         a.sort()
36         print (a)
37
38
39
40
Line 19, Column 1 Spaces: 4 Python
```

Code for user interface:

```
C:\Users\rohit54\Desktop\nlp\New folder\project1.py • - Sublime Text 2 (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
search.py x test.py x grade.py project1.py abc.py x
1 from tkinter import *
2 root = Tk()
3 root.title("My Application")
4 root.geometry("1300x700+0+0")
5
6 def trying():
7     iv = quest.get()
8     print(iv)
9 head = Label(root, bd=2, text="Question-Answer System", font=("Times",35, "bold"), fg="blue").pack()
10
11 ques = Label(root, bd=2, text="Question:",font=("Times",25,"bold"), fg="red").place(x=20, y=100)
12 ans1 = Label(root, bd=2, text="Answer 1:",font=("Times",25,"bold"), fg="green").place(x=20, y=200)
13 ans2 = Label(root, bd=2, text="Answer 2:",font=("Times",25,"bold"), fg="green").place(x=20, y=300)
14 ans3 = Label(root, bd=2, text="Answer 3:",font=("Times",25,"bold"), fg="green").place(x=20, y=400)
15 ans4 = Label(root, bd=2, text="Answer 4:",font=("Times",25,"bold"), fg="green").place(x=20, y=500)
16 ans5 = Label(root, bd=2, text="Answer 5:",font=("Times",25,"bold"), fg="green").place(x=20, y=600)
17 #ans6 = Label(root, bd=2, text="Answer 6:",font=("Times",25,"bold"), fg="green").place(x=20, y=630)
18
19 quest = StringVar()
20 answ1 = StringVar()
21 answ2 = StringVar()
22 answ3 = StringVar()
23 answ4 = StringVar()
24 answ5 = StringVar()
25 #answ6 = StringVar()
26
27 ent_box1 = Text(root, bd=2, height=3, width=80, font=("arial",12,"bold")).place(x=180, y=100)
28 #ent_box1.pack()
29 ent_box2 = Text(root, bd=2, height=3, width=80, font=("arial",12,"bold")).place(x=180, y=200)
30 ent_box3 = Text(root, bd=2, height=3, width=80, font=("arial",12,"bold")).place(x=180, y=300)
31 ent_box4 = Text(root, bd=2, height=3, width=80, font=("arial",12,"bold")).place(x=180, y=400)
32 ent_box5 = Text(root, bd=2, height=3, width=80, font=("arial",12,"bold")).place(x=180, y=500)
33 ent_box6 = Text(root, bd=2, height=3, width=80, font=("arial",12,"bold")).place(x=180, y=600)
34 #ent_box7 = Entry(root, bd=2, textvariable=answ6, width=40, font=("arial",12)).place(x=180, y=510)
35 btn1=Button(root, height=9, bd=3, width=30,text="COSINE SIMILARITY MEASURE",font=("Times",10,"bold"),command=trying).place(x=950,y=500)
36
37 btn2=Button(root, height=9,bd=3, width=30,text="EUCLIDEAN SIMILARITY MEASURE",font=("Times",10,"bold"),command=trying).place(x=950,y=300)
38
39 btn3=Button(root, height=9,bd=3, width=22,text="INSERT",font=("Times",10,"bold"),command=trying).place(x=950,y=100)
40
41 mainloop()
42
43
44
45
Line 35, Column 1 Spaces: 4 Python
```