Name: Akshay Surendra Phadnis
Group-name: AKSHAY_PHADNIS
UTD-ID: 2021256400
NET-ID: asp150630

Project Description

Problem Description:

The objective of this project is to enhance the query generation task (using Natural Language Processing) for a closed domain question –answering system (QA system) that uses unstructured text as a knowledge-base. Generating the right queries to be fired against unstructured text is of significant moment in order to extract the right passages to be ranked and thus, to find an answer closest to the expected answer. Query generation is used in a QA system where it has been highlighted below.

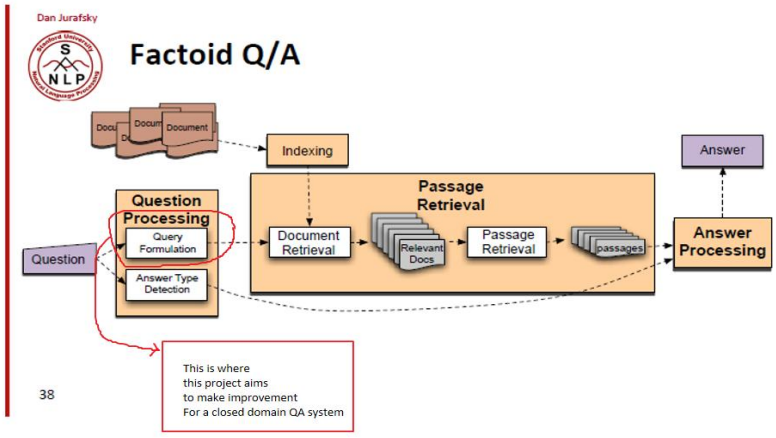


Image taken from slides by Prof. Dan Jurafsky, Stanford University

The task of query generation entailing choosing words from a given question as keywords is important because the data i.e. the corpus against which the query will be fired are not going to adhere to a fixed format. They'll be available as text files. Thus, the problem is: how to choose words from a question that will facilitate finding the correct answer in a corpus of substantial size.

Proposed Solution and Implementation Details

Baseline system

A naïve approach would be to choose words randomly.
Example:
Consider the task of generating queries for a closed domain QA system that answers questions regarding C programming.

Suppose the QA system gets the following question as input:
How to access the element of a structure via its pointer?

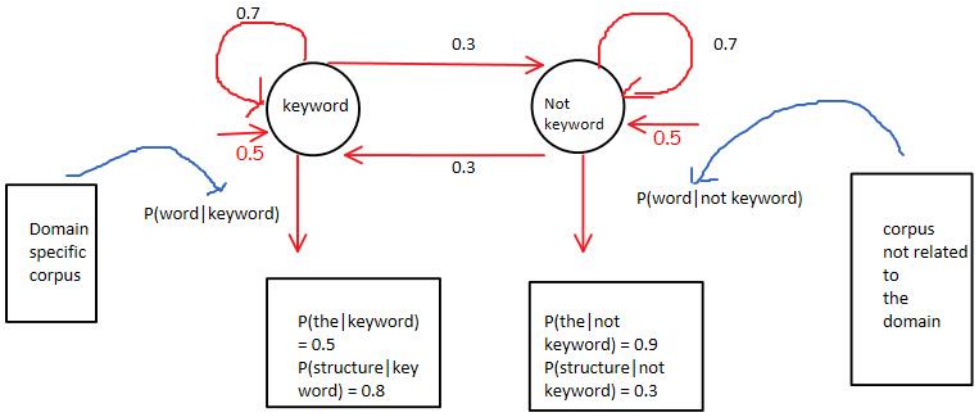
Then, disparate queries as shown by the highlighted words below will be obtained on every run of the program:

How	to	access	the	element	of	a	structure	via	it's	pointer	?

How	to	access	the	element	of	a	structure	via	it's	pointer	?

Improvement strategy
A significant enhancement would be to choose words based on a probability model. The task at hand has been perceived as follows:

For each word in the question, we want to decide whether or not it is a keyword. This can be considered as a decoding problem similar to tagging, where for each word, the tag can be either keyword or not-keyword. A Hidden Markov Model as shown below has been used:



The emission probabilities have been computed using two different corpora as delineated by the figure above and are maximum between probability of a word and it's stem.

Once the model was established, the Viterbi algorithm was is for decoding. The eventual result of this algorithm is a sequence of tags (keyword or not) for each word in the question.

Examples
For the same question, a tag sequence similar to the following will be generated:

How	to	access	the	element	of	a	structure	via	it's	pointer	?
NK	NK	K	NK	K	NK	NK	K	NK	NK	K	NK

NK denotes not a keyword
K denotes keyword

The result from Viterbi might still not be perfect thus, part of speech tagging is used to divest the question of the words falsely tagged as keywords i.e. words corresponding to some tags which cannot be keywords .Some examples of such tags are:

- Pronouns
- Wh-determiner
- Determiner
- The word to
- Existential there
- Verb, non-3rd person singular present
- Verb, 3rd person singular present

The query (which is a collection of words) after POS, is then verified for its relevance with the related corpus using Lesk algorithm (Word Sense Disambiguation), if relevant, is returned. This is important because the related corpus might contain words outside of the domain and the number of such words in the related corpus can be greater than the number of same words in the unrelated corpus. To invalidate such words as keywords, this approach was taken.
For instance, when C programming is the closed domain and the query 'your birthday' is returned after POS, these words will be considered irrelevant to C programming corpus by Lesk algorithm and will be discarded.

Programming tools

Programming Languages used:

Python 2.7

External Libraries used:

Natural Language Toolkit (NLTK) for part of speech tagging and stemming

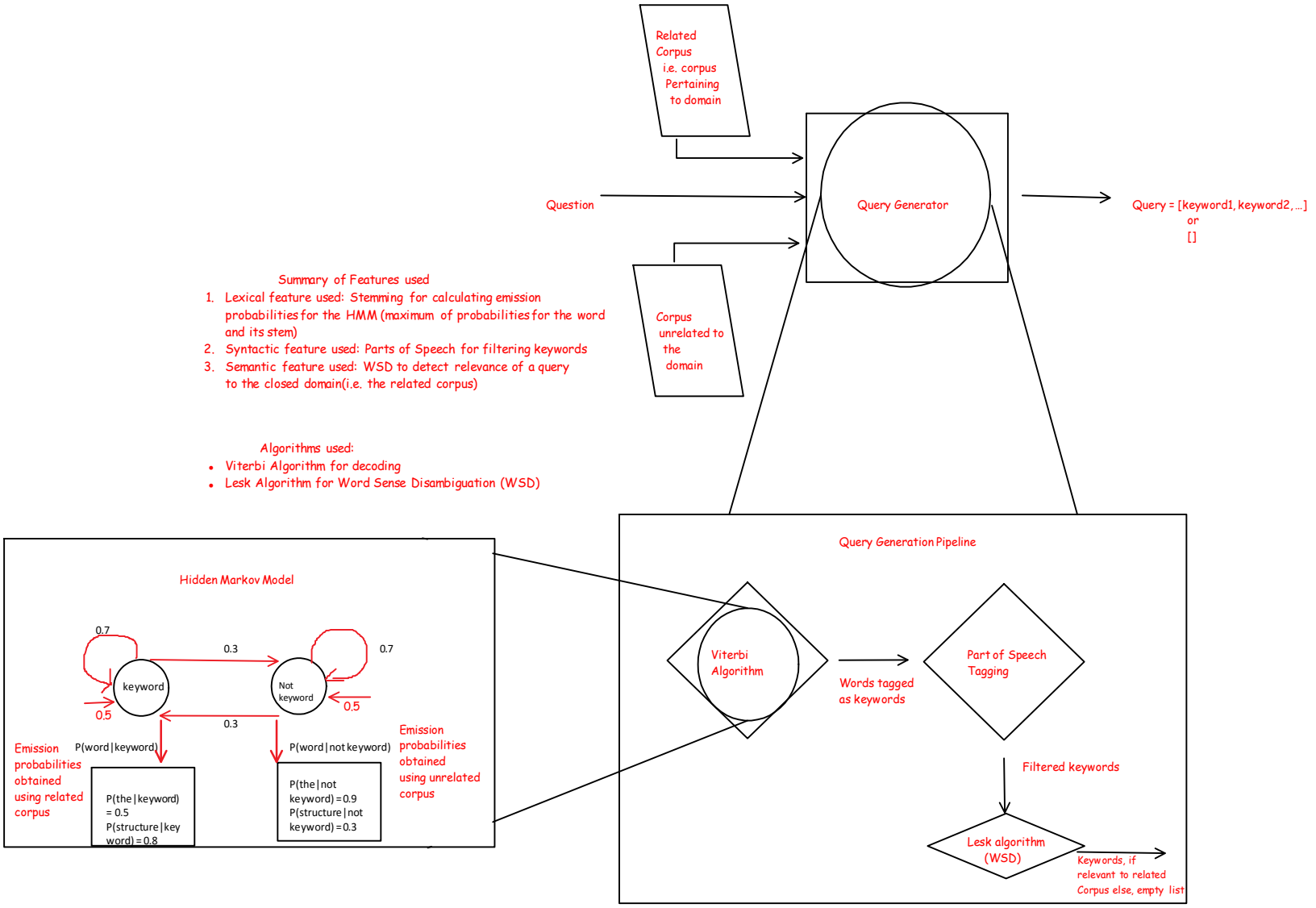
Related Corpus:

Collection of articles from Wikipedia, openculture.com, M.I.T. OCW, etc. related to C programming

Unrelated Corpus:

Collection of freely available English literature like 'Three men in a Boat', 'The Adventures of Sherlock Holmes', etc. From Project Guttenberg (https://www.gutenberg.org/)

Architectural diagram



Results:

Implementation of the above pipeline yielded an accuracy of about 89%. Following are outputs to some questions and each output shows emission probabilities from related and unrelated corpus (with entries K and NK respectively), output of Part of Speech Tagging and the final query obtained after checking relevance to the related corpus(C programming is the closed domain under consideration):

QUESTION

how do I access the element of a structure via its pointer
{('a', 'K'): 0.02168911658687204,
 ('a', 'NK'): 0.018824211742910792,
 ('access', 'K'): 0.0003214429807374465,
 ('access', 'NK'): 6.78627297888e-05,
 ('do', 'K'): 0.001159845806784601,
 ('do', 'NK'): 0.00161401358373981,
 ('element', 'K'): 0.0006834805647123541,
 ('element', 'NK'): 2.3437128329121648e-05,
 ('how', 'K'): 0.0011971265648598201,
 ('how', 'NK'): 0.001069712513887373,
 ('i', 'K'): 0.00283747992016947,
 ('i', 'NK'): 0.007894464360173393,
 ('its', 'K'): 0.006288849656644218,
 ('its', 'NK'): 0.009385695376099446,
 ('of', 'K'): 0.019660214886289544,
 ('of', 'NK'): 0.030854104925574874,
 ('pointer', 'K'): 0.0027206668782004495,
 ('pointer', 'NK'): 0.00018434875566338967,
 ('structure', 'K'): 0.0012310934777727978,
 ('structure', 'NK'): 2.8334438726251543e-05,
 ('the', 'K'): 0.051880731398765756,
 ('the', 'NK'): 0.06028833964386759,
 ('via', 'K'): 5.8820751629790474e-05,
 ('via', 'NK'): 4.897310397129896e-06}
Tagged query:
[('access', 'NN'),
 ('the', 'DT'),
 ('element', 'NN'),
 ('of', 'IN'),
 ('a', 'DT'),
 ('structure', 'NN'),
 ('via', 'IN'),
 ('its', 'PRP$'),
 ('pointer', 'NN')]
For the question
how do i access the element of a structure via its pointer

words related to C programming are i.e. the query words are
['access', 'element', 'structure', 'pointer']

QUESTION
How many storage classes exist in C

{('c', 'K'): 0.004741283965877336,
 ('c', 'NK'): 0.00011683583376009895,
 ('classes', 'K'): 0.00029244683556783153,
 ('classes', 'NK'): 9.374851331648659e-05,
 ('exist', 'K'): 0.00011929842584070181,

('exist', 'NK'): 2.6935207184214428e-05,
('how', 'K'): 0.0011971265648598201,
('how', 'NK'): 0.001069712513887373,
('in', 'K'): 0.012773216177860697,
('in', 'NK'): 0.0173578170868559,
('many', 'K'): 0.0005616967549999711,
('many', 'NK'): 0.0007024142341026308,
('storage', 'K'): 0.00031315836783184225,
('storage', 'NK'): 1.7490394275463916e-06}
Tagged query:
[('how', 'WRB'),
('many', 'JJ'),
('storage', 'NN'),
('classes', 'NNS'),
('exist', 'VBP'),
('in', 'IN'),
('c', 'NN')]
For the question
how many storage classes exist in c

words related to C programming are i.e. the query words are
['storage', 'classes', 'c']

QUESTION
is there a built in function to compute sin

{('a', 'K'): 0.02168911658687204,
('a', 'NK'): 0.018824211742910792,
('built', 'K'): 4.970767743362575e-05,
('built', 'NK'): 7.345965595694844e-05,
('compute', 'K'): 8.036074518436163e-05,
('compute', 'NK'): 6.996157710185566e-07,
('function', 'K'): 0.004913603914313906,
('function', 'NK'): 2.518616775666804e-05,
('in', 'K'): 0.012773216177860697,
('in', 'NK'): 0.0173578170868559,
('is', 'K'): 0.016613134259608288,
('is', 'NK'): 0.008702170767814316,
('sin', 'K'): 3.4795374203538024e-05,
('sin', 'NK'): 3.1482709695835046e-05,
('there', 'K'): 0.0012667173132668963,
('there', 'NK'): 0.002477339445176709,
('to', 'K'): 0.02049696078975558,
('to', 'NK'): 0.02341054292982294}
Tagged query:
[('is', 'VBZ'),
('there', 'EX'),
('a', 'DT'),
('built', 'NN'),
('in', 'IN'),
('function', 'NN'),
('to', 'TO'),

('compute', 'VB'),
('sin', 'NN')]
For the question
is there a built in function to compute sin

words related to C programming are i.e. the query words are
['built', 'function', 'compute', 'sin']

QUESTION
I must say, with deepest regret that I do not know how to obtain square root of a number using c

{('a', 'K'): 0.02168911658687204,
('a', 'NK'): 0.018824211742910792,
('c', 'K'): 0.004741283965877336,
('c', 'NK'): 0.00011683583376009895,
('deepest', 'K'): 2e-19,
('deepest', 'NK'): 1.1543660221806184e-05,
('do', 'K'): 0.001159845806784601,
('do', 'NK'): 0.00161401358373981,
('how', 'K'): 0.0011971265648598201,
('how', 'NK'): 0.001069712513887373,
('i', 'K'): 0.00283747992016947,
('i', 'NK'): 0.007894464360173393,
('know', 'K'): 0.000357066816231545,
('know', 'NK'): 0.0010053478629536657,
('must', 'K'): 0.0009750989389896252,
('must', 'NK'): 0.0008860633739950019,
('not', 'K'): 0.0028325091524261073,
('not', 'NK'): 0.005402782791690803,
('number', 'K'): 0.002004876323156239,
('number', 'NK'): 0.0002581582195058474,
('obtain', 'K'): 5.7163829048669614e-05,
('obtain', 'NK'): 3.987809894805773e-05,
('of', 'K'): 0.019660214886289544,
('of', 'NK'): 0.030854104925574874,
('regret', 'K'): 2e-19,
('regret', 'NK'): 2.1338281016065975e-05,
('root', 'K'): 0.00011267073551621837,
('root', 'NK'): 1.9589241588519584e-05,
('say,', 'K'): 2e-19,
('say,', 'NK'): 5.5969261681484525e-06,
('square', 'K'): 0.00020960070651178858,
('square', 'NK'): 5.422022225393814e-05,
('that', 'K'): 0.007986366841002538,
('that', 'NK'): 0.009531915072242325,
('to', 'K'): 0.02049696078975558,
('to', 'NK'): 0.02341054292982294,
('using', 'K'): 0.0016088718262683534,
('using', 'NK'): 0.0007734252348610143,
('with', 'K'): 0.003941818820486522,
('with', 'NK'): 0.007439014493240312}
Tagged query:

[('obtain', 'VB'),
 ('square', 'JJ'),
 ('root', 'NN'),
 ('of', 'IN'),
 ('a', 'DT'),
 ('number', 'NN'),
 ('using', 'VBG'),
 ('c', 'NNS')]
For the question
i must say, with deepest regret that i do not know how to obtain square root of a number using c

words related to C programming are i.e. the query words are
['obtain', 'square', 'root', 'number', 'using', 'c']

QUESTION
C programming just added to the vississitudes of life I just do not know if there is a built in function for sorting

{('a', 'K'): 0.02168911658687204,
 ('a', 'NK'): 0.018824211742910792,
 ('added', 'K'): 0.0002269983936135576,
 ('added', 'NK'): 0.00018504837143440824,
 ('built', 'K'): 4.970767743362575e-05,
 ('built', 'NK'): 7.345965595694844e-05,
 ('c', 'K'): 0.004741283965877336,
 ('c', 'NK'): 0.00011683583376009895,
 ('do', 'K'): 0.001159845806784601,
 ('do', 'NK'): 0.00161401358373981,
 ('for', 'K'): 0.00739153163438015,
 ('for', 'NK'): 0.006112542991389129,
 ('function', 'K'): 0.004913603914313906,
 ('function', 'NK'): 2.518616775666804e-05,
 ('i', 'K'): 0.00283747992016947,
 ('i', 'NK'): 0.007894464360173393,
 ('if', 'K'): 0.004963311591747532,
 ('if', 'NK'): 0.00218105216615035,
 ('in', 'K'): 0.012773216177860697,
 ('in', 'NK'): 0.0173578170868559,
 ('is', 'K'): 0.016613134259608288,
 ('is', 'NK'): 0.008702170767814316,
 ('just', 'K'): 0.0007804105357079244,
 ('just', 'NK'): 0.0007797217768001813,
 ('know', 'K'): 0.000357066816231545,
 ('know', 'NK'): 0.0010053478629536657,
 ('life', 'K'): 4.142306452802146e-05,
 ('life', 'NK'): 0.0007688777323493938,
 ('not', 'K'): 0.0028325091524261073,
 ('not', 'NK'): 0.005402782791690803,
 ('of', 'K'): 0.019660214886289544,
 ('of', 'NK'): 0.030854104925574874,
 ('programming', 'K'): 0.00378358271398948,
 ('programming', 'NK'): 1.4691931191389689e-05,
 ('sorting', 'K'): 0.0002750491484660625,

('sorting', 'NK'): 0.00013257718860801648,
('the', 'K'): 0.051880731398765756,
('the', 'NK'): 0.06028833964386759,
('there', 'K'): 0.0012667173132668963,
('there', 'NK'): 0.002477339445176709,
('to', 'K'): 0.02049696078975558,
('to', 'NK'): 0.02341054292982294,
('vississitudes', 'K'): 2e-19,
('vississitudes', 'NK'): 2e-10}
Tagged query:
[('c', 'JJ'),
('programming', 'NN'),
('just', 'RB'),
('added', 'VBN'),
('if', 'IN'),
('there', 'EX'),
('is', 'VBZ'),
('a', 'DT'),
('built', 'VBN'),
('in', 'IN'),
('function', 'NN'),
('for', 'IN'),
('sorting', 'VBG')]
For the question
c programming just added to the vississitudes of life i just do not know if there is a built in function for sorting

words related to C programming are i.e. the query words are
['c', 'programming', 'function', 'sorting']

QUESTION
When is your birthday

{('birthday', 'K'): 1.076999677728558e-05,
('birthday', 'NK'): 1.7840202160973195e-05,
('is', 'K'): 0.016613134259608288,
('is', 'NK'): 0.008702170767814316,
('when', 'K'): 0.0019634532586282174,
('when', 'NK'): 0.0023919863211124453,
('your', 'K'): 0.001426610342345059,
('your', 'NK'): 0.0013481595907527586}
Tagged query:
[('when', 'WRB'), ('is', 'VBZ'), ('your', 'PRP$'), ('birthday', 'NN')]
For the question
when is your birthday

words related to C programming are i.e. the query words are
[]

QUESTION
Give me a list of Norse gods

{('a', 'K'): 0.02168911658687204,

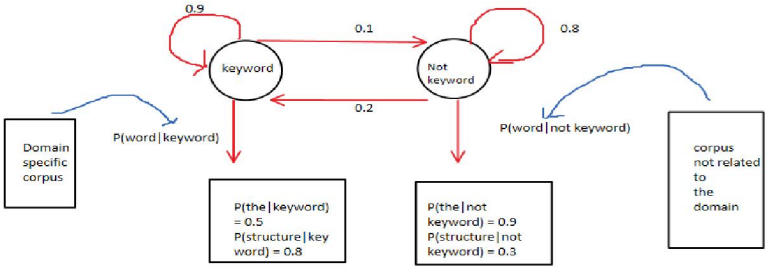
('a', 'NK'): 0.018824211742910792,
('give', 'K'): 0.00017977610005161313,
('give', 'NK'): 0.00047818737949118347,
('gods', 'K'): 6.627690324483434e-06,
('gods', 'NK'): 0.00032077383101200823,
('list', 'K'): 0.0008367459034660335,
('list', 'NK'): 4.442560145967834e-05,
('me', 'K'): 6.710536453539477e-05,
('me', 'NK'): 0.0018819664240399172,
('norse', 'K'): 2e-19,
('norse', 'NK'): 3.498078855092783e-07,
('of', 'K'): 0.019660214886289544,
('of', 'NK'): 0.030854104925574874}
Tagged query:
[('a', 'DT'), ('list', 'NN')]
For the question
give me a list of norse gods

words related to C programming are i.e. the query words are
['list']

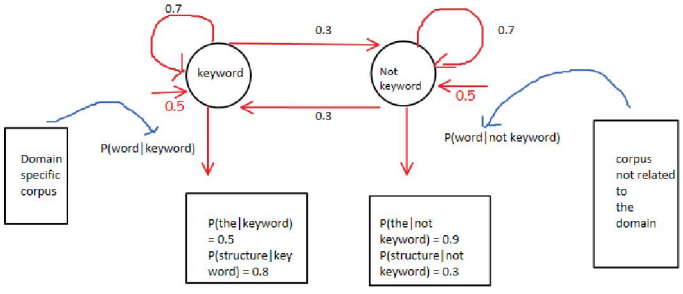
Problems encountered and their resolutions

Transition probabilities for the HMM:

These probabilities initially were very high for remaining in the same state as delineated by the figure (initial state probabilities are 0.5 and 0.5) below:



Due to this, if a word was tagged as K, Viterbi algorithm in an attempt to maximize joint probability of words and tags preferred 0.9 or 0.8 over 0.1 or 0.2 and as a result, sequences of Ks or NKs was obtained. To alleviate this problem, these probabilities were reduced as can be seen in the currently used model below:



Emission probabilities for words that do not appear in corpora:

If some words do not appear in the corpus they will be assigned an emission probability of zero. But, doing so would cause the joint probability to become zero thus, a very low probability has been assigned to such words.

Also, a preference had to be set for such words i.e. for a word not appearing in any of the two corpora, it has to be assigned either a K or an NK. In order to reduce any possible false positives, such words are being treated as NKs and this is being ensured by keeping 'Not keyword' state's emission probabilities for such words more than their 'keyword' state's emission probabilities.

Augmentation of corpus:

Initially, the related corpus contained more number of certain words which are supposed to be non-keywords than the unrelated corpus. For instance, the word 'what' appeared more number of times in the related corpus. Due to this emission probabilities for such words was higher for 'keyword' state. To resolve such issues with words like 'who', 'how', etc. the unrelated corpus was augmented with a list of questions.

Unimportant words getting tagged as keywords:

Some unimportant words (like 'how', 'the', 'a', etc.) were however tagged as keywords sometimes. To resolve this issue, from the partial query (obtained after performing decoding using Viterbi algorithm) words with part of speech tags DT, WRB, PRP$, JJ, etc. were removed.

Keywords getting tagged as non-keywords:

Due to filtering effect of the above mentioned Part of Speech Tagging, adjectives like 'square' in 'how to find square root of a number in c ?' and 'double' in 'how to convert double to an int ?' were not being considered as keywords. This issue was resolved by giving special treatment to words with tags JJ and VBG. These words are now filtered out only when e[word,'K'] < e[word,'NK'] , where e represents emission probability.

Distinctively non-keywords being tagged as non-keywords:

Some words like 'apple' and 'juice' appeared more number of times in related corpus than in the unrelated corpus, even after corpus augmentation. Due to this queries like ['make', 'apple', 'juice'] were generated (for the question 'How to make apple juice ?') This issue was resolved by using Lesk algorithm for Word Sense Disambiguation. Lesk algorithm essentially checks if a string i.e. the filtered query (obtained after part of speech tagging) is relevant (has an intersection of greater size) to the related corpus.

Effect of passing synonyms of question words along with question to the query generation pipeline

In order to improve the project, instead of just considering words appearing in the question as the input, synonyms of these words (obtained using sysnets from WordNet) were also considered after doing part of speech tagging on the question. This approach increases the time complexity of the program due to proliferation of words in the question.

Potential improvements

Ontologies specific to the related corpus can be made and synonyms of question words from these ontologies can be used while generating emission probabilities (maximum among probabilities of synonyms can be used as the emission probability).