

A semester project

report on

Summarization of Medical Blogs-

A probe on the performance of multiple techniques that
can be employed for the purpose

Submitted by:

Prem Kamal (BE-CSE/1130/2011)

Saurabh Garg (BE-CSE/1146/2011)

Rahul Nori (BE-CSE/1159/2011)

(Group No – 29)



Under the guidance of:

Dr. Sujan Kumar Saha

Assistant Professor

BIT Mesra

Date of Submission: 30th April 2015

ABSTRACT

Online doctor is a term that emerged during 2000s, used by both the media academics, to describe a generation of physicians and health practitioners who deliver healthcare, including drug prescription, over the internet.

The patients describe their problem in the form of a paragraph or a detailed passage on the website. Online doctors go through these descriptions and provide a suitable solution.

This project focuses on summarizing the results of a blog written on the online medical websites in a few words, to detect the possible symptom or a set of possible symptoms. Some of the benefits of this project are that the patient receives a quicker reply and time spent on reading descriptions by doctors is reduced. It will also ensure that a doctor of a specific field does not end up spending his precious time on reading descriptions of problems of another genre. Consequently, doctors won't refrain from replying in case of large sized written blogs.

CERTIFICATE

This is to certify that the project report entitled Summarization of Medical Blogs submitted by Prem Kamal (BE/1130/2011), Saurabh Garg (BE/1146/2011) & Rahul Nori (BE/1159/2011) to BIT Mesra is a record of bonafide work carried out by them under my supervision and guidance in the session 2014-15.

(Dr. S.K.Saha)

BIT Mesra

Project Guide

ACKNOWLEDGEMENTS

Firstly, we would like to express our sincere gratitude to our project supervisor Dr. Sujan Kumar Saha, Assistant Professor, BIT Mesra, who guided us, all through the project and showed us the right path. We are very thankful that he has spent so much time with us during my project work.

My honourable thanks to Dr. Sandip Dutta, Associate Professor and Head of Department of CSE, BIT Mesra for providing excellent labs and library.

We thank BIT Mesra for providing us with necessary infrastructure and technical support that was required for this project.

Finally, we express our gratitude to our parents and friends for their valuable suggestions and moral support.

Prem Kamal (BE/1130/2011)

Saurabh Garg (BE/1146/2011)

Rahul Nori (BE/1159/2011)

BIT Mesra

CONTENTS

1. Introduction
 - 1.1 Online Doctoring
 - 1.2 Current Scenario
 - 1.3 Problems associated with Online Doctoring
 - 1.4 Motivation and Objectives
2. Literature Survey
3. Data Collection
 - 3.1 Dictionary of words
 - 3.2 List of symptoms
 - 3.3 Sample data
4. Algorithms
 - 4.1 Spell Checking Algorithm
 - 4.2 Algorithm to find the root word of all the words in the description
 - 4.3 Extracting all the symptoms from the input
 - 4.4 Rule-based extraction
 - 4.5 Using MALLET
5. Design and Implementations
 - 5.1 Dictionary Extractor
 - 5.2 Symptoms List Extractor
 - 5.3 Spell Checker (Code cleaner)
 - 5.4 Root Word Converter
 - 5.5 Symptoms Extractor from Input
 - 5.6 Rule Based Extractor
 - 5.7 Extraction using MALLET
6. Comparative Study of the Three Algorithms

Conclusions and Future Work

References

Chapter 1: Introduction

1.1 Online Doctoring

Online doctor is a term that emerged during the 2000s, used by both the media and academics, to describe a generation of physicians and health practitioners who deliver healthcare, including drug prescription, over the internet.

Health advice is now the second-most popular topic that people search for on the internet. With the advent of broadband and videoconferencing, many individuals have turned to online doctors to receive online consultations and purchase prescription drugs.

Advantages of Online Doctoring –

1. Cost savings and convenient.
2. Accessibility and improved privacy and communication.
3. Patients can consult licensed physicians online – from anywhere, at any moment.
4. It allows doctors to give immediate online health tips and advice in rural areas through the internet broadband and videoconferencing system.

1.2 Current Scenario

There are hundreds of medical websites that offer the patients online medical facilities. Examples include:

1. <https://www.doctorspring.com>
2. <https://www.icliniccare.com>
3. <https://www.icliniq.com>

The patients describe their problem in the form of a paragraph or a detailed passage on the website. This can be done 24 hours a day, 7 days a week, be it any remote location. Also, the patients can attach the required prior diagnosis from other doctors. There are many online doctors who go through the descriptions and provide a solution in their free time. These doctors who are available on the online websites are certified in their field of expertise.

1.3 Problems associated with online doctoring

Most of the problems associated with online doctoring are a result of the descriptions given by the patients. Some of the problems are:

1. The doctors are usually busy attending to their offline clients and cannot spend much time reading the lengthy descriptions.
2. The patient describing his/her problem may not be proficient with the language in which they are posting the description and hence there may be many errors. The doctors, being busy, may not be able to spend much time deciphering or correcting the mistakes.
3. Many a times, a doctor ends up reading the complete description and then realizes that the disease the patient is describing is not of his genre which results in unnecessary usage of his/her time.

1.4 Motivation and Objectives

The purpose of the project is to summarize the results of a written blog on the online medical websites in a few words, to detect the possible symptom or a set of possible symptoms.

This will lead to following advantages:

1. A quicker reply to the patient.
2. A doctor's time spent on reading the whole description is reduced.
3. Doctors won't refrain from replying in case of large sized written blogs.
4. A doctor of a specific field doesn't end up spending his precious time on reading descriptions of problems of another genre.

Goal is to make a full fledged system for doctors to get a quick review of the problem description given by the patients on various online medical sites.

The main objectives to implement the goal are –

Method 1 –

1. Create a database of the disease and its related symptoms.
2. Cleaning the description provided by the user.
3. Deriving the root words from the cleaned text.
4. Finding out the symptoms from the information derived.

Method 2 –

1. Create a set of rules to extract symptoms from the text.
2. Cleaning the description provided by the user.
3. Matching the text against the set of rules.

Method 3 –

1. Create a training model by taking a few texts and manually specifying each symptom in each of them.
2. Prepare a testing set and running it on MALLET to get a list of symptoms.

Chapter 2: Literature Survey

The first medical consulting website in the US was WebMD, founded in 1996 by Jim Clark (one of the founders of Netscape). Currently, its website carries information regarding health and health care, including a symptom checklist, pharmacy information, drug information, blogs of physicians with specific topics, and a place to store personal medical information.

In the UK, e-med was the first online health site to offer both a diagnosis and prescriptions to patients over the Internet. It was established in March 2000 by Dr. Julian Eden.

The project requires the study of stemming algorithms and spell checking techniques. Alternatively, we require basic knowledge of regular expressions and pattern matching algorithms in order to identify symptoms efficiently and effectively. We could also use MALLET for Sequence Tagging and training the model to get the expected results.

The first published stemmer was written by Julie Beth Lovins in 1968.

A later stemmer was written by Martin Porter and was published in the July 1980 issue of the journal *Program*. This stemmer was very widely used and became the de facto standard algorithm used for English stemming.

Ralph Gorin, a graduate student under Earnest at the time, created the first true spelling checker program written as an applications program (rather than research) for general English text: Spell for the DEC PDP-10 at Stanford University's Artificial Intelligence Laboratory, in February 1971. Gorin wrote SPELL in assembly language, for faster action; he made the first spelling corrector by searching the word list for plausible correct spellings that differ by a single letter or adjacent letter transpositions and presenting them to the user.

A regular expression is a sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations. The concept arose in the 1950s, when the American mathematician Stephen Kleene formalized the description of a *regular language*, and came into common use with the Unix text processing utilities [ed](#), an editor, and global regular expression print, a filter.

MALLET was developed primarily by Andrew McCallum, of the University of Massachusetts Amherst, with assistance from graduate students and faculty from both UMASS and the University of Pennsylvania. MALLET is an integrated collection of Java code useful for statistical natural language processing, document classification, cluster analysis, information extraction, topic modeling and other machine learning applications to text.

In this chapter, literature survey on spell checking techniques and stemming algorithms, rule-based extraction and NLP techniques is presented.

1. Spell Checking Techniques in NLP: A Survey – International Journal of Advanced Research in Computer Science and Software Engineering, Neha Gupta, Pratishtha Mathur, Volume 2, Issue 12, December 2012.

1. Explained approaches and roles of spell checkers in various applications.
2. Explained the types of errors and error detection and correction methods.

2. A Comparison of Standard Spell Checking Algorithms and a Novel Binary Neural Approach – Victoria J. Hodge and Jim Austin – Volume 15 – No. 5 – 2003 – IEEE Transactions on Knowledge and Data Engineering.

1. Compared the standard spell checking algorithms.

3. A Comparative Study of Stemming Algorithms – Anjali Ganesh Jivani et al, Volume 2(6), International Journal of Computer Technology and Applications.

1. Explained stemming and lemmatization.
2. Explained different methods of stemming and their comparisons.

4. Classification and Rule Extraction using Rough Set for Diagnosis of Liver Disease and its Types – S.Karthik, A Priyadarshini, J Anuradha, B.K.Tripathy, Advances in Applied Science Research, 2011, 2(3):334-345

1. Gave an idea how Regular Expressions can be used in symptom extraction.

5. Using Machine Learning for Medical document Summarization – Kamal Sarkar, Mita Nasipuri, Suranjan Ghose, International Journal of Database Theory and Application, Vol. 4, No. 1, March 2011.

1. Explained how machine learning can be used for medical document summarization.

6. Medical Information Extraction Using Natural language Interpretation – Gunjan Dhole, Dr. Nilesh Uke, Advances in Vision Computing, Vol. 1, No. 1, March 2014.

1. Suggested a model consisting of two modules: Document Processing and Query Processing with Natural Language Processing.

7. NLP Based Retrieval of Medical Information for Diagnosis of Human Diseases – Gunjan Dhole, Nilesh Uke, International Journal of Research in Engineering and Technology. 2319-1163. pISSN: 2321-7308

1. Explains the way to diagnose diseases with the help of natural language interpretation and classification techniques.

2. Implements the model.

8. MapFace – An Editor for MetaMap Transfer (MMTx). Katharina Kaiser, Theresia Gschwandtner, Patrick Martini. Vienna University of Technology.

1. Explained the use of MetaMap designed by UMLS.

Chapter 3: Data Collection

3.1 Dictionary of words:

A list of words is created to implement the spell checker program.

The words are extracted from the website –

<http://www-personal.umich.edu/~jlawler/wordlist>

The total number of words in the list is 69,903.

Technologies used in extraction are JAVA (Regular Expressions, File Read/Write Modules and URL package)

Algorithm –

Regex matching to find the pattern in the source code of the website.

3.2 List of symptoms:

Database consists of all the symptoms which is needed to extract information from the input given by the patient.

The required information is extracted from the following websites –

1. <http://www.medicinenet.com/>
2. <http://symptomchecker.webmd.com>

Technologies used are – Python 2.7, urllib, urllib2, bs4

Algorithm –

The pattern is found out in the page source of the website and the contents are derived by separating the tags.

3.3 Sample Data

The inputs given by the patients can be extracted from the websites:

1. <https://onlinedoctor.lloydspharmacy.com/>
2. <https://www.lloydonlinedoctor.ie/>
3. <http://www.netdoctor.co.uk>

Technologies Used – Python

Algorithm –

Cron Job / Timed requests by the doctor.

Chapter 4: Algorithms

4.1 Spell Checking Algorithm –

The misspelt or erratic words are to be replaced with the most suitable word in the context. Statistical based approach can be used to find the most suitable word for replacement. The list of all English words created previously will be used for this purpose.

Algorithm –

1. If the word is present in the dictionary, directly return the word.
2. If it is not, then choose the one that requires the least number of changes to form the possible word. The changes include: insertion / deletion / replacement / transposition of characters and splitting of word, all given equal weights.
3. In case of ambiguity, the terms that are medically more suitable are printed out as they have the maximum candidature value.

4.2 Algorithm to find the root word of all the words in the description –

Porter's algorithm is used to find out the suitable root word by eliminating the possible verb forms. It follows the following rules. Once a word has been formed, we find it in the dictionary to check if it exists or the closest word that exists for its replacement.

Rule			Example		
SS	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

4.3 Extracting all the symptoms from the input –

Algorithm–

1. Check every root word from the description if it is available in the database of symptoms. This check is made only when the word has not already been marked.
2. If such a word is found, it is marked and added to symptoms set.
3. A map is used to ensure that a single symptom is not marked and counted more than once.

Technologies Used – Python + List of symptoms created previously

4.4 Rule Based Extraction –

A set of rules is created using regular expressions (Regex) and the code matches the rules against the sample text to extract symptoms.

Few sample rules are as follows:

Rule	Example
[a-z0-9]* y(ea)?rs old	21 years old
[a-z0-9]* y(ea)?r old	six year old
experiencing [^\.]*	experiencing chest pain
experienced [^\.]*	experienced pain in legs
had [^\.]*	had headache
have [^\.]*	have swelling
having [^\.]*	having high fever
has [^\.]*	has been vomiting
got [^\.]*	toes got dry
get [^\.]*	get itchy

there is [a-z]* in [^\.]*	there is swelling in legs
became [^\.]*	became nauseous
noticed [^\.]*	noticed lump behind a knee
found [^\.]*	found white patches
developed [^\.]*	developed a medical condition
feeling [^\.]*	stiffness
felt [^\.]*	weakness
suffering from [^\.]*	suffering from body ache
[a-z]* hurts	arm hurts
[a-z]* is hurting	right hand is hurting
treated for [^\.]*	treated for cancer
operated for [^\.]*	operated for lung cancer
[a-z]* has led to [^\.]*	dryness has led to cut
symptoms are [^\.]*	symptoms are sore wrists
problem is [^\.]*	problem is irritation in right eye

4.5 Using MALLET

SimpleTagger is a command line interface to the MALLET Conditional Random Field (CRF) Class.

The input file should be in the following format –

Daughter #NA

has #NA

sores #SB

in #NA

her #NA

mouth. #NA

She #NA

also #NA

has #NA

flu-like #SB

symptoms. #SC

The three classes are:

#NA – Not Applicable. #SB – Symptom Begins. #SC – Symptom Continues.

Each line represents a token and has the format:

feature1 feature2 featureN label

Then, the CRF is trained using SimpleTagger like this :

```
java -cp class;lib\mallet-deps.jar cc.mallet.fst.SimpleTagger --train true --model-file nouncrf sample_new.txt
```

The --train true option specifies that we are training, and --model-file nouncrf specifies where we would like the CRF written to.

The output of a test case is obtained like this:

```
java -cp class;lib\mallet-deps.jar cc.mallet.fst.SimpleTagger --model-file nouncrf test_new.txt
```

Chapter 5: Design and Implementation

The design and implementation of the important code snippets are presented below:

1. Dictionary Extractor

Design:

Using regular expression matching, the words have been scraped out of the online dictionary file

Code Snippet:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.regex.*;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class Dic
{

    public static void main(String[] args) throws IOException
    {

        String inputLine,s="";
        int a[]={19,9,21,10,10,9,8,10,9,1,2,10,15,8,6,23,1,9,21,12,4,7,2,1,1,1};
        Pattern patt = Pattern.compile("<td><li><a href=. *[0-9]\\ ">(. *)(</a></li></td>");

        File file = new File("C:\\lab\\fms\\Dictionary.txt");
        FileWriter fw = new FileWriter(file.getAbsolutePath());
        BufferedWriter bw = new BufferedWriter(fw);

        for(int i=97;i<=122;i++)
        {

            String p="http://www.medilexicon.com/medicaldictionary.php?l=";
            char q=(char)i;
            String h= p+q;
            URL x = new URL(h);
            BufferedReader in = new BufferedReader(new InputStreamReader(x.openStream()));
            while ((inputLine = in.readLine()) != null)
            {
```

```

        Matcher m = patt.matcher(inputLine);
while (m.find()) {
    bw.write(m.group(2)+"\n");
}

    }
    in.close();
    if(a[i-97]>1)
    {
        for(int j=2;j<=a[i-97];j++)
        {
            p="http://www.medilexicon.com/medicaldictionary.php?l=";
            q=(char)i;
            h= p+q+"&s=&p="+Integer.toString(j);
            URL x2 = new URL(h);
            BufferedReader in2 = new BufferedReader(new InputStreamReader(x2.openStream()));
while ((inputLine = in2.readLine()) != null)
        {
            Matcher m2 = patt.matcher(inputLine);
while (m2.find()) {
                bw.write(m2.group(2)+"\n");
            }
        }
        in.close();
    }
}

bw.close();

}
}

```

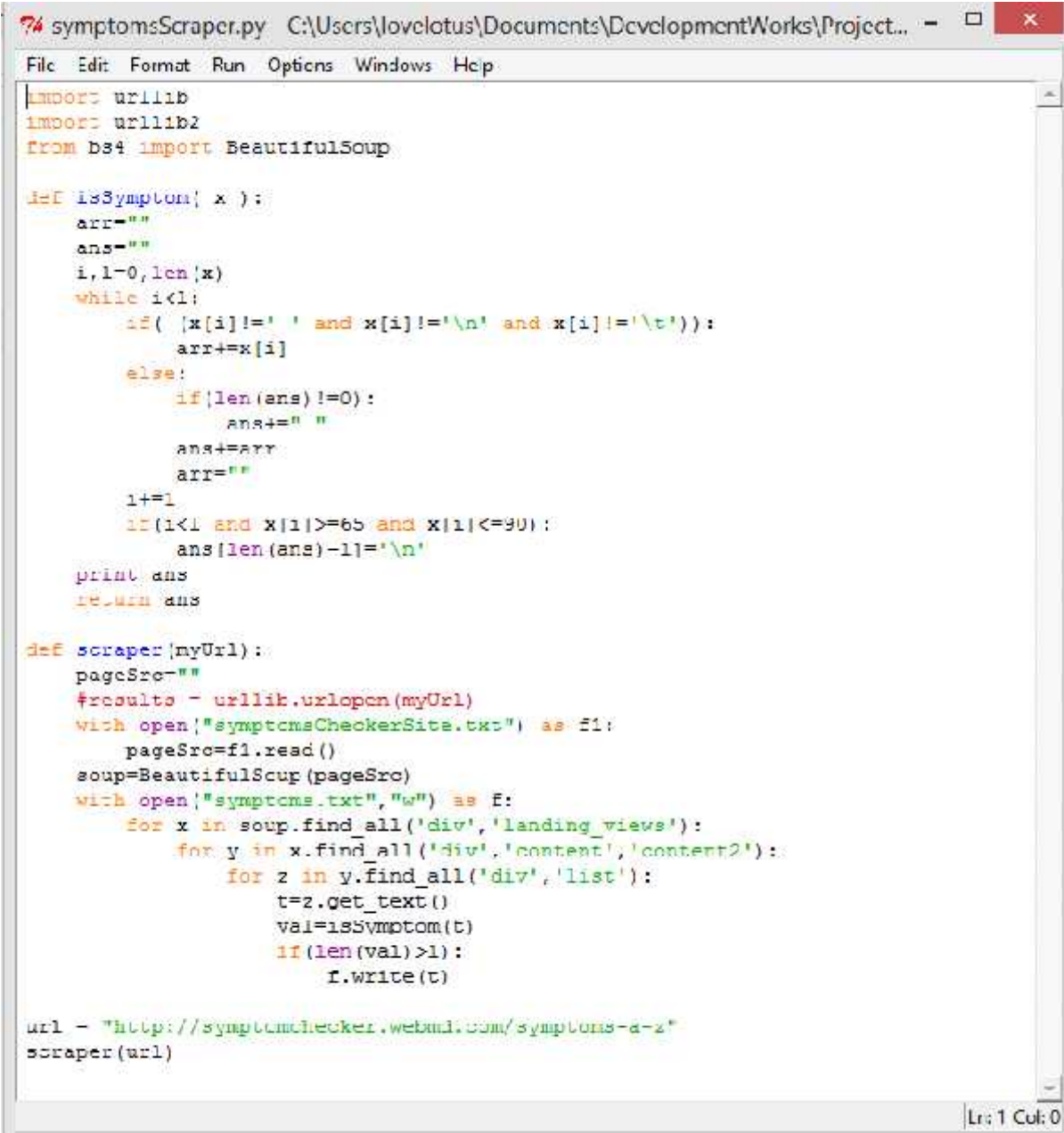
(Coded in Java)

2. Symptoms List Extractor

Design:

The symptoms have been extracted from the 'symptomschecker.net' website using a BeautifulSoup module, which lets the developer identify the pattern of the different tags and enables the user to find and extract the suitable data.

Code Snippet:



```
symptomsScrapper.py C:\Users\lovelotus\Documents\DevelopmentWorks\Project... - □ ×
File Edit Format Run Options Windows Help
import urllib
import urllib2
from bs4 import BeautifulSoup

def isSymptom( x ):
    arr=""
    ans=""
    i,l=0,len(x)
    while i<l:
        if ( x[i]!=' ' and x[i]!='\n' and x[i]!='\t' ):
            arr+=x[i]
        else:
            if(len(ans)!=0):
                ans+=" "
            ans+=arr
            arr=""
        i+=1
        if(i<l and x[i]>=65 and x[i]<=90):
            ans[len(ans)-1]='\n'
    print ans
    return ans

def scraper(myUrl):
    pageSrc=""
    #results = urllib.urlopen(myUrl)
    with open("symptomsCheckerSite.txt") as f1:
        pageSrc=f1.read()
    soup=BeautifulSoup(pageSrc)
    with open("symptoms.txt","w") as f:
        for x in soup.find_all('div','landing_views'):
            for y in x.find_all('div','content','content2'):
                for z in y.find_all('div','list'):
                    t=z.get_text()
                    val=isSymptom(t)
                    if(len(val)>1):
                        f.write(t)

url = "http://symptomschecker.webmd.com/symptoms-a-z"
scraper(url)
```

Ln: 1 Col: 0

3. Spell Checker (Code cleaner)

Design:

The spell checker algorithm uses a predefined dictionary, with the correct words as input and increased frequency of the words, those are more likely to be present in form of symptoms. The algorithm has been described already in section 4.1

Code Snippet:

```
spellChecker.py - C:\Users\lovekshus\Documents\Development\Works\Project\metaphyspellChecker.py

File Edit Format Run Options Windows Help

import re
myDict = {}

def replacement1(word):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    splits = [(word[i:i+1], word[i+1:i+2]) for i in range(len(word)-1)]
    posChange1 = [a + b[1] for a, b in splits] # making deletions
    posChange2 = [a + b[1] + b[0] for a, b in splits] # making transposes
    posChange3 = [a + c + b[1] for a, b in splits for c in alphabet] # making replacements
    posChange4 = [a + c + b for a, b in splits for c in alphabet] # making insertions
    return set(posChange1 + posChange2 + posChange3 + posChange4)

def replacement2(word):
    return set([a + b[1] + b[0] for a, b in replacement1(word)] + [a + b[1] + b[0] for a, b in splits])

def known(words):
    return set(w for w in words if w in myDict)

def present(word):
    if word in myDict:
        return 1
    else:
        return 0

def correct(word):
    word = word.lower()
    if present(word):
        return word
    else:
        candidates = known([word]) or known(replacement1(word)) or replacement2(word) or [word]
        return min(candidates, key=lambda x: len(x)-len(word))

def search(text):
    return re.findall('[a-z]+', text.lower())

def buildIndex(words):
    tempDict = collections.defaultdict(lambda: 0)
    for i in words:
        tempDict[i] += 1
    return tempDict

myDict = normDictionary(wordsFile, 'words.txt', read())

def run():
    text = ""
    with open("StopInput.txt") as f:
        text = f.read()
        #print text
    cleanCode = ""
    temp = ""
    for x in text:
        if x != ' ' and x != '\n' and x != '\t' and x != '\r':
            if len(temp) > 0:
                word = correct(temp)
                cleanCode += word
                #print temp, word
                cleanCode += " "
                temp = ""
            else:
                temp = x
        elif len(temp) > 0:
            cleanCode += correct(temp)
            #print cleanCode
    with open("StopOutput.txt", "w") as f:
        f.write(cleanCode)

run()
```

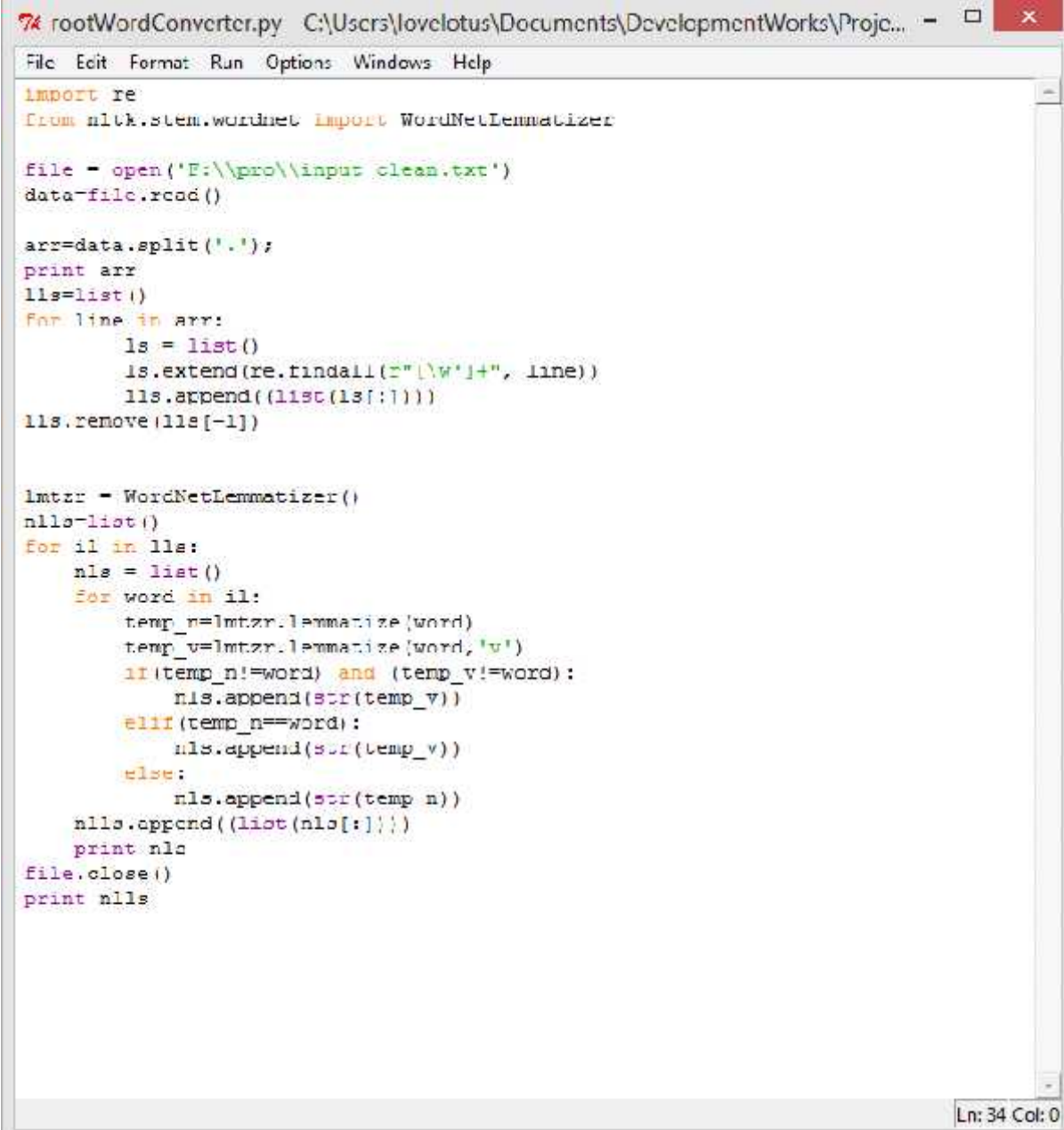
4. Root Word Converter

Design:

The first step is reduction to words according to the Porter's algorithm

The next step is to lemmatize the words to find the word that is closest to it in the dictionary.

Code Snippet:



```
rootWordConverter.py C:\Users\lovelotus\Documents\DevelopmentWorks\Projc... - □ ×
File Edit Format Run Options Windows Help

import re
from nltk.stem.wordnet import WordNetLemmatizer

file = open('E:\\pro\\input_clean.txt')
data=file.read()

arr=data.split('.');
print arr
l1s=list()
for line in arr:
    l1 = list()
    l1.extend(re.findall(r"[\w']+", line))
    l1s.append((list(l1[:]))))
l1s.remove(l1s[-1])

lmtzr = WordNetLemmatizer()
n1s=list()
for il in l1s:
    n1s = list()
    for word in il:
        temp_n=lmtzr.lemmatize(word)
        temp_v=lmtzr.lemmatize(word, 'v')
        if (temp_n!=word) and (temp_v!=word):
            n1s.append(str(temp_v))
        elif (temp_n==word):
            n1s.append(str(temp_v))
        else:
            n1s.append(str(temp_n))
    n1s.append((list(n1s[:]))))
    print n1s
file.close()
print n1s

Ln: 34 Col: 0
```

5. Symptoms Extractor from Input

Design:

Each symptom is searched in the complete input file for its presence. If the presence is found, the symptom is mapped, to ensure that it is not repeated again.

Code Snippet:



```
74 symptom_extractor.py - C:\Users\lovelotus\Documents\DevelopmentWorks\Proje...
File Edit Format Run Options Windows Help

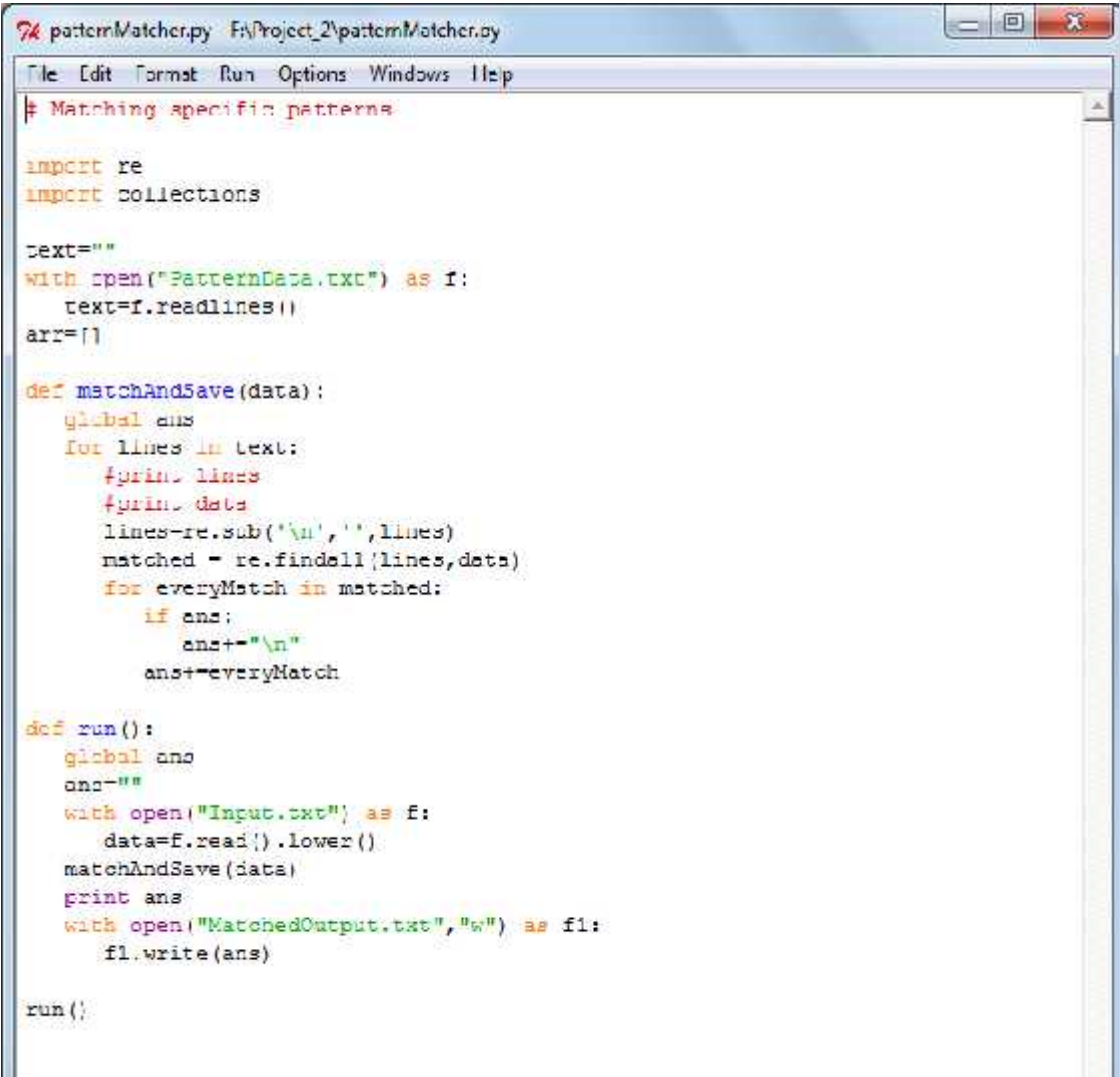
import re
file = open('Step2Input.txt')
data1=file.read()
arr1=data1.split('.')
#print arr1
l1s1=list()
for line in arr1:
    ls = list()
    ls.extend(re.findall(r"[\w']+", line))
    l1s1.append(list(ls[:]))
l1s1.remove(l1s1[-1])
#print l1s1
file.close()
file = open('Symptoms_nltk.txt')
data2=file.read()
arr2=data2.split('\n')
#print arr2
l1s2=list()
for line in arr2:
    | ls = list()
    ls.extend(re.findall(r"[\w']+", line))
    l1s2.append(list(ls[:]))
l1s2.remove(l1s2[-1])
#print l1s2
file.close()
line_num = list()
for lis2 in l1s2:
    for lis1 in l1s1:
        ctr=0
        for x in lis2:
            for y in lis1:
                if x==y:
                    #print x,y
                    ctr=ctr+1
                    break
            if ctr==len(lis2):
                ind = l1s2.index(lis2)
                print ind
                line_num.append(ind)
file = open('Symptoms.txt')
data3=file.read()
arr3=data3.split('\n')
file.close()
#print arr3
dic = {}
for ind in line_num:
    #print arr3[ind]
    dic[ind] = arr3[ind]
#print dic
st=''
with open("extractedSymptoms.txt","w") as fl:
    for key in dic:
        st=st+dic[key]
        st = st + "\n"
        fl.write(st)
    arr='
```

6. Rule-based Extraction

Design:

Each rule is matched against the sample input and the matched items are written to a text file.

Code Snippet:



```
patternMatcher.py  F:\Project_2\patternMatcher.py
File Edit Format Run Options Windows Help
# Matching specific patterns

import re
import collections

text=""
with open("PatternData.txt") as f:
    text=f.readlines()
arr=[]

def matchAndSave(data):
    global ans
    for lines in text:
        #print lines
        #print data
        lines=re.sub('\n',' ',lines)
        matched = re.findall(lines,data)
        for everyMatch in matched:
            if ans:
                ans+="-\n"
            ans+=everyMatch

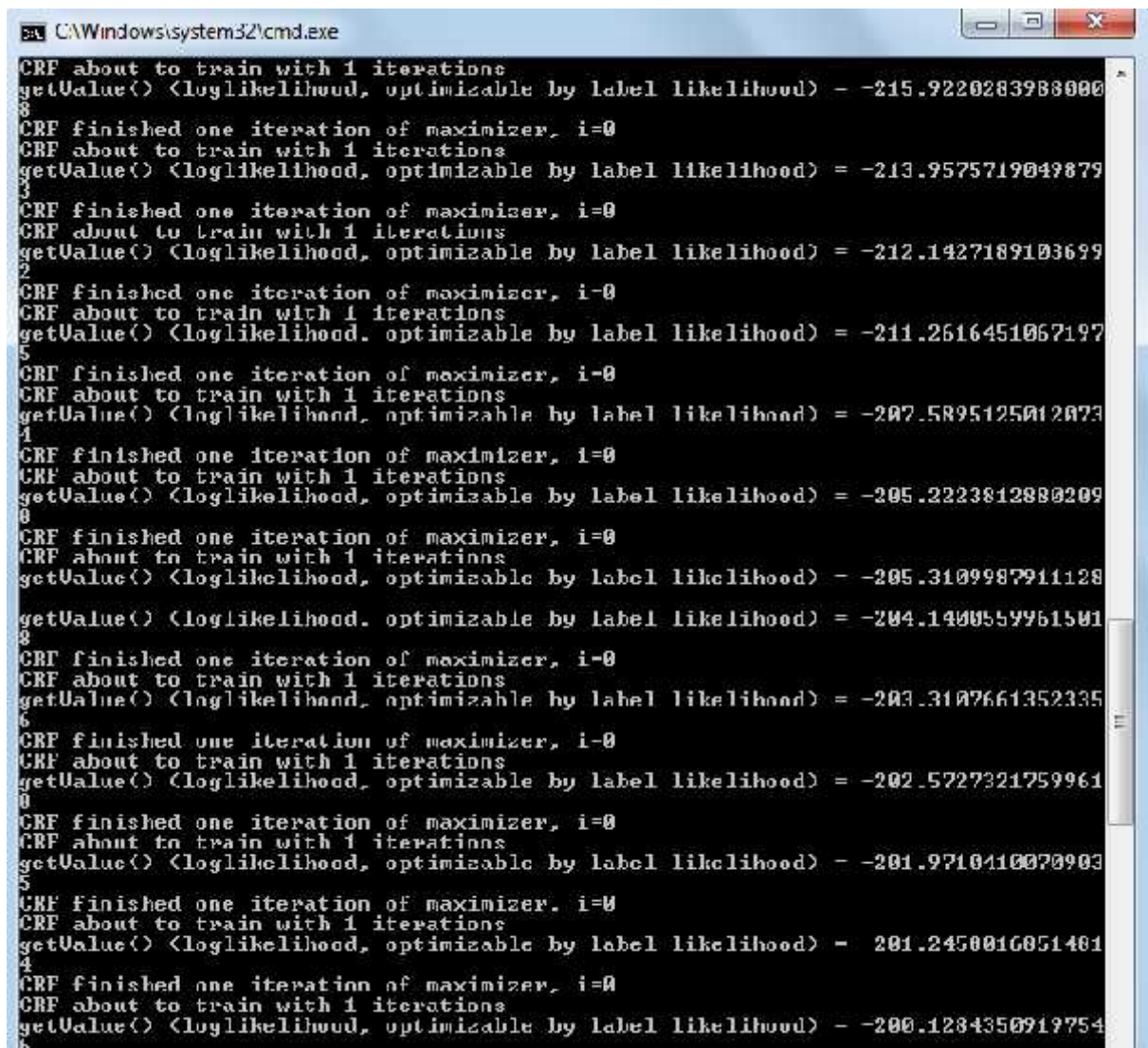
def run():
    global ans
    ans=""
    with open("Input.txt") as f:
        data=f.read().lower()
    matchAndSave(data)
    print ans
    with open("MatchedOutput.txt","w") as f1:
        f1.write(ans)

run()
```


7. Using MALLET:

Training:

```
java -cp class;lib\mallet-deps.jar cc.mallet.fst.SimpleTagger --train true --  
model-file nouncrf sample_new.txt
```



```
CAWindows\system32\cmd.exe  
CRF about to train with 1 iterations  
getValue() <loglikelihood, optimizable by label likelihood> = -215.9220283988000  
8  
CRF finished one iteration of maximizer, i=0  
CRF about to train with 1 iterations  
getValue() <loglikelihood, optimizable by label likelihood> = -213.9575719049879  
3  
CRF finished one iteration of maximizer, i=0  
CRF about to train with 1 iterations  
getValue() <loglikelihood, optimizable by label likelihood> = -212.1427189103699  
2  
CRF finished one iteration of maximizer, i=0  
CRF about to train with 1 iterations  
getValue() <loglikelihood, optimizable by label likelihood> = -211.2616451067197  
5  
CRF finished one iteration of maximizer, i=0  
CRF about to train with 1 iterations  
getValue() <loglikelihood, optimizable by label likelihood> = -207.5895125012073  
1  
CRF finished one iteration of maximizer, i=0  
CRF about to train with 1 iterations  
getValue() <loglikelihood, optimizable by label likelihood> = -205.2223812880209  
0  
CRF finished one iteration of maximizer, i=0  
CRF about to train with 1 iterations  
getValue() <loglikelihood, optimizable by label likelihood> = -205.3109987911128  
8  
CRF finished one iteration of maximizer, i=0  
CRF about to train with 1 iterations  
getValue() <loglikelihood, optimizable by label likelihood> = -203.3107661352335  
6  
CRF finished one iteration of maximizer, i=0  
CRF about to train with 1 iterations  
getValue() <loglikelihood, optimizable by label likelihood> = -202.5727321759961  
0  
CRF finished one iteration of maximizer, i=0  
CRF about to train with 1 iterations  
getValue() <loglikelihood, optimizable by label likelihood> = -201.9710410070903  
5  
CRF finished one iteration of maximizer, i=0  
CRF about to train with 1 iterations  
getValue() <loglikelihood, optimizable by label likelihood> = -201.2450016051401  
4  
CRF finished one iteration of maximizer, i=0  
CRF about to train with 1 iterations  
getValue() <loglikelihood, optimizable by label likelihood> = -200.1284350919754  
6
```

Testing:

```
java -cp class;lib\mallet-deps.jar cc.mallet.fst.SimpleTagger --model-file  
nouncrf test_new.txt
```

Chapter 6: Comparative Study of Three Algorithms

The three algorithms were tested on a number of test cases. The results varied on the basis of the input data set provided. The result derived varied with the change in data as described below:

1. The results of the dictionary based approach varied with the change in the number and variations of words provided in the dictionary.
2. The results derived from the rule-based approach varied with the number and pattern of rules.
3. The results derived from the algorithms including hidden Markov models (HMMs) and linear chain conditional random fields (CRFs), implemented using Mallet.

The performance comparison can be done with the help of the following example:

For a small input as :

```
1 My wife had a headache for 3 days. Today she became nauseous on the way home after lunch.  
2 She has now vomited six times. She has ache in joints.  
3 She has no strength. Continued to feel as if hit by bowling ball just below the chest.  
4 What can I do to help her?  
5
```

The results derived from the various techniques were:

1. Dictionary based approach

```
1 No strength  
2 Joint aches  
3 Headache
```

2. Rule based approach

```
1 had a headache for 3 days
2 has now vomited six times
3 has ache in joints
4 has no strength
5 became nauseous on the way home after lunch
```

3. Learning based approach

```
1 headache for 3 days
2 vomited six times
3 ache in joints
4 no strength
5 nauseous
```

From the results observed above, it can be said that the rule-based and learning based approaches work better than dictionary based approach in a general case when the rules and training set are very well defined. But the fact that needs to be considered is the finiteness of the number of rules and training set data.

For the rules based approach, the number of rules that can be defined is limited by the memory size of the system as well as the running size of the system. Hence, working practically, the results that will be obtained will work very accurately for the data that is related to the rules while on the other hand, it may extract very small result for a content that is far different from the rules design.

For the learning based approach, the training data must be large and variant in nature so that it can classify the data precisely and provide more accurate results. Still the training set required less memory and running time than that of the rules-based approach.

From the results derived, we can determine that the results derived from the learning based approach would work more efficiently and effectively for a larger data set than the other two approaches and hence is more favorable than the other two algorithms employed.

Conclusions and Future Work

Firstly, online doctoring, its advantages and disadvantages and the current scenario are discussed. Then, the problems faced by the online doctors are explained in detail.

Secondly, the process of how the databases (used in different algorithms) have been constructed has been explained.

Then the data extracted from the medical blogs website was cleaned using a self-devised spell checking algorithm.

The next step involved three different algorithms implementation. Three algorithms have been developed to create a list of symptoms from the input given by the patients. For the achievement of this objective, firstly, a dictionary based approach has been employed that used the modified version of the Porter's algorithm to find the best possible root word and match it against the root word found out from the symptoms-dictionary. The second algorithm involved a rule-based approach that used a predefined rules-pattern to match and extract data. The third algorithm involved a learning based approach in which initially the system was trained on a data set and then the symptoms were extracted from the data-set based on the training.

The final conclusion drawn was that learning-based approach is most suitable for practical usage. For an extremely large data for which training cannot be sufficiently provided (i.e. there is a restriction on memory), the dictionary based approach might work comparably but learning-based approach works good for all of them.

In the future, the possible disease or a set of possible diseases based on the symptoms extracted shall be detected using machine learning concepts or fuzzy logic.

References

- Spell Checking Techniques in NLP: A Survey – International Journal of Advanced Research in Computer Science and Software Engineering, Neha Gupta, Pratishtha Mathur, Volume 2, Issue 12, December 2012.
- A Comparison of Standard Spell Checking Algorithms and a Novel Binary Neural Approach – Victoria J. Hodge and Jim Austin – Volume 15 – No. 5 – 2003 – IEEE Transactions on Knowledge and Data Engineering.
- Spell Checking – http://en.wikipedia.org/wiki/Spell_checker
- Stemming - <http://en.wikipedia.org/wiki/Stemming>
- Porter's Algorithm - <http://snowball.tartarus.org/algorithms/porter/stemmer.html>
- A Comparative Study of Stemming Algorithms – Anjali Ganesh Jivani et al, Volume 2(6), International Journal of Computer Technology and Applications.
- Classification and Rule Extraction using Rough Set for Diagnosis of Liver Disease and its Types – S.Karthik, A Priyadarshini, J Anuradha, B.K.Tripathy, Advances in Applied Science Research, 2011, 2(3):334-345
- Using Machine Learning for Medical document Summarization – Kamal Sarkar, Mita Nasipuri, Suranjan Ghose, International Journal of Database Theory and Application, Vol. 4, No. 1, March 2011.
- Medical Information Extraction Using Natural language Interpretation – Gunjan Dhole, Dr. Nilesh Uke, Advances in Vision Computing, Vol. 1, No. 1, March 2014.
- MapFace – An Editor for MetaMap Transfer (MMTx). Katharina Kaiser, Theresia Gschwandtner, Patrick Martini. Vienna University of Technology.

- NLP Based Retrieval of Medical Information for Diagnosis of Human Diseases – Gunjan Dhole, Nilesh Uke, International Journal of Research in Engineering and Technology. 2319-1163. pISSN: 2321-7308.
- MetaMap – A Tool for Recognizing UMLS Concepts in Text - <http://metamap.nlm.nih.gov/>
- Mallet - <http://mallet.cs.umass.edu/>