IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727
PP 72-79
www.iosrjournals.org

NLP Based Event Extraction from Text Messages

Aneesh G. Nath, Krishnanth V, Kevin Biju Mathew, Pranav T S, Sarath Gopi
Department of Computer Science &Engg.,T K M College of Engg., Karicode, Kollam-691005, Kerala, India.

Abstract: *Natural language processing has made its mark in many of the applications recently released. The process which helps in communication and handling natural language by machines has led to exploration in the field of computer science and machine learning. In this paper, we propose an application which extracts events from text messages and adds them to the calendar and notify on specific time. The text message is initially treated by a lexical analyzer and split into tokens and these tokens are taken in as the input to the POS tagging phase. After proper tagging it's provided to the parser. A parser is provide with the grammar which is developed after examining a set of messages and its general trend. The parser constructs the parse tree according to the grammar from which we obtain the relevant parts of the message that is event, date, time and place. After successful extraction these are mapped on to the calendar by using a cross platform. Since the natural language processing does better in python than other languages and converting to an application is done by a cross platform, Kiwi, which helps to convert python code to java code. Since java code is better for android the application works well.*

Keywords: *Tokenizing; Grammar; Parser; Event Extraction; Kivy; Buildozer*

I. Introduction

In the present world common people receive a lot of text messages as a notification for them about certain events and people do add them to calendar in order to notify as if the numbers of such messages are large. People follow various methods like keeping a note on them in a book, adding the event to the calendar manually so that they will notify on the day even if we forget about it. Adding to calendar makes alerts on the mobile based on the event and meeting date.

Since the world is so hectic, manually adding data to a calendar after checking message requires a lot of time. Messages are of many types: Personal messages, official, notification type etc. The main domain of our concern is the messages which arrive with an event in it. It includes messages having a date, time, event, and place or either of these present which indicates an event.

Our project aims at automating the process of extraction of event and adding it on to the calendar. We classify the arriving messages into events, date, time and place if any of the combination is present and making use of natural language processing. We takes the event and maps on to the calendar with the appropriate date provided in the message. Thus the main concentration is on 4 major tasks.

- Subject extraction
- Date extraction
- Time extraction
- Place extraction.

The extraction is based on the grammar. A grammar includes certain rules which are to be considered to include and exclude words based on the tags. NLP plays a key role in this process. An application was developed in order to implement this by integrating python code with android. A number of test cases were considered which led is us to find common pattern in the messages and so that define a grammar. Grammar helps us to construct a tree and this tree will result into various extractions. Thus we get our event mapped on to the calendar and notify the user at right time.

II. Literature Survey

On a daily basis, most email users will send and receive numerous emails of varying types thereby generating massive amounts of digital communication. At present, it is the user's responsibility to filter and archive all of her email data which, depending upon the average number of messages she receives, may be an extremely time consuming task. Often important information is not properly archived or noted and, as a result, lost in a sea of email communication. The problem of event extraction from incoming message has been approached in the past by numerous research efforts. Julie A. Blackand, NisheethRanjan of Stanford University proposed a system for Automated Event Extraction from Email[1].To mitigate the problems associated with

event extraction from emails, they proposed an architecture for automated event extraction from incoming email messages. A successful system will properly classify messages that contain event information, attempt to perform information extraction to isolate the specifics of the event (date, time, location, title, and participants), present all of this information to the user for confirmation. Three types of email were defined as follows:
Official meeting emails are all messages that contain event information clearly delimited from all other text in the email. Most frequently, seminar announcements are presented in this format and would fall into this category. *Personal meeting emails* are meeting proposals in which the specifics of the meeting are presented within the body of the email itself. We term such emails as personal meetings as this format is most commonly used for friends and acquaintances to propose meetings.*Other emails* are all incoming messages that we deem to be non-event related.

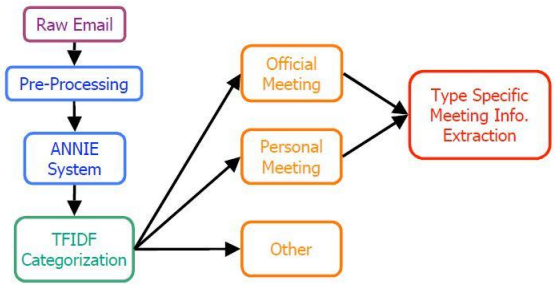

Fig 1: architecture of information extraction system

The raw email data is exported to a single text file where personal, official, and otheremails are demarcated by XML tags. This single email file is then processed by a Perl (pre-process.pl) and separated out into three separate files containing personal meeting emails, official meeting emails, and other emails respectively. All three output files are passed through another Perl script (strip-quoted.pl) that strips all quoted text from the emails. We do this because we only want to find event information in the real body of the email message, not in the entire thread of the email conversation contained within the email.

1.1 ANNIE

Developed at the University of Sheffield, ANNIE – a Nearly New Information Extraction
System – was used as a named entity recognizer after the pre-processing step. ANNIE helps to provide tags of people, locations, dates, parts of speech, and sentence boundaries. These tags appear in the outputted body of the raw email as XML tags wrapping the recognized text. Later, these tags are relied upon for additional information about the email. ANNIE provides a tokenizer, a gazetteer, a sentence splitter, a part of speech tagger, a semantic tagger, an orthographic coreferencer, and a pronominal corefrencer. These components can be used independently of one another to provide different annotations required at different points in the system developed.
The ANNIE pipeline used was composed of the tokenizer, the gazetteer, the sentence splitter, the part of speech tagger, and the named entity transducer.

1.2 TF-IDF-based Categorization

In order to categorize different emails into the three categories a similarity measure based on TF-IDF (Term Frequency – Inverse Document Frequency) was used. In addition to this similarity measure, numerous domain specific heuristics with hand-tuned weights were defined. This is described in more detail below.

1.3 Information Extraction

In order to extract information from the categorized email messages, the RAPIER algorithm proposed by Califf and Mooney was used. RAPIER uses pairs of sample documents and filled templates to induce pattern match rules that directly extract fillers for the slots in the template. It employs a bottom up learning algorithm which incorporates techniques from several inductive logic programming systems to learn patterns that include constraints on the words and part of speech tags present in the filler and surrounding text.

1.4 RAPIER (Robust Automated Production of IE Rules)

RAPIER's learning program learns rules that use constraints on words and on part of speech tags. It trains on a training set where each training example is a collection of three files: (a) email.orig, the original email, (b) email_out, the original email after passing it through a sentence splitter and part of speech tagger, (c)

email.template, a filled event template containing the date, time, location, and title of the event contained in the email. ANNIE's sentence splitter and POS tagger export their output as verbose XML files where the POS tags were contained in attributes on <token> tags around each text token created by the ANNIE tokenizer. RAPIER expected email_out to be in the format of "word/tag" where 'tag' is the POS tag of 'word'.

1.5 Perl based Pattern Matching

In addition to the RAPIER system discussed briefly above, an information extraction pattern matcher which relies upon the information returned by ANNIE to complement the raw email data when extracting the specifics of a meeting or event from an incoming email message. To simplify the problem, the focus was only on extracting only the date, location and title of the event using a few key features.

They weren't able to achieve the performance necessary for deploying the information extraction system as a plugin to an email client.

Appointment information extraction from short messages

In 2011 Choong-Nyoung Seon and team proposed a system for appointment information extraction from short messages in mobile devices with limited hardware resources[3].The proposed system extracts temporal (dates and times) and named instances (locations and title) from Korean short messages in an appointment management domain. To efficiently extract temporal instances with limited numbers of surface forms, the proposed system uses well-refined finite state automata. To effectively extract various surface forms of named instances with limited hardware resources, the proposed system uses a modified hidden Markov model (HMM) based on character n-grams.

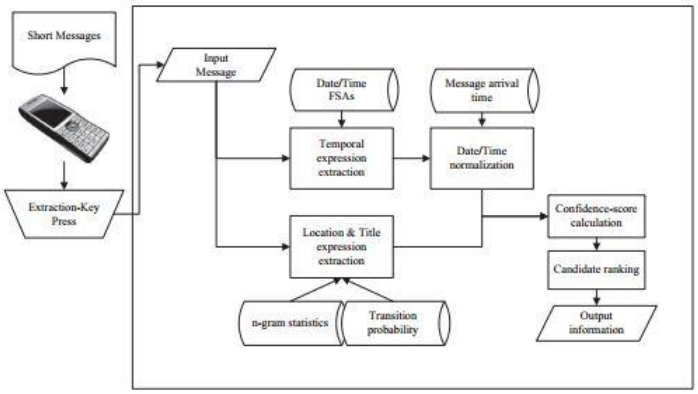

Fig 2: The overall architecture of Seon's system

In the temporal instance extraction part, the proposed system extracts temporal instance candidates (i.e. dates and times) using the well-known FSA. Then, the system converts the temporal instances into machine-manageable forms. In the named entity extraction part, the proposed system extracts named instance candidates (i.e. locations and the title) using a modified HMM based on character n-grams.

Temporal instance extraction using FSA Although short messages in an appointment domain often include many incorrect words, temporal instances like dates and times are usually expressed correctly because they are very important in appointment messages. In addition, temporal instances are expressed in tractable numbers of surface forms in order to allow message receivers to easily understand.

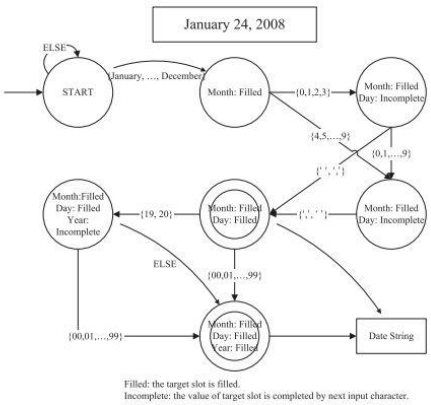

Fig 3: Temporal instance using FSA

Named entity extraction was done using a modified HMM and decision trees. Unlike dates and times, locations and titles not only have various surface forms, but their constituent words are also not included in a closed set. The proposed system efficiently extracted temporal instances with limited numbers of surface forms by using FSA. To effectively extract various surface forms of named instances with small hardware resources, the proposed system used a modified HMM based upon character n-grams.

III. Proposed Working Of The Application

The whole purpose of the project is to determine whether an event details are specified in a text message. If details are present the map the details to calendar. The input for the system is the text message received by the user in his/her phone and the final output is a calendar event. There are different stages of the project as shown in the figure.

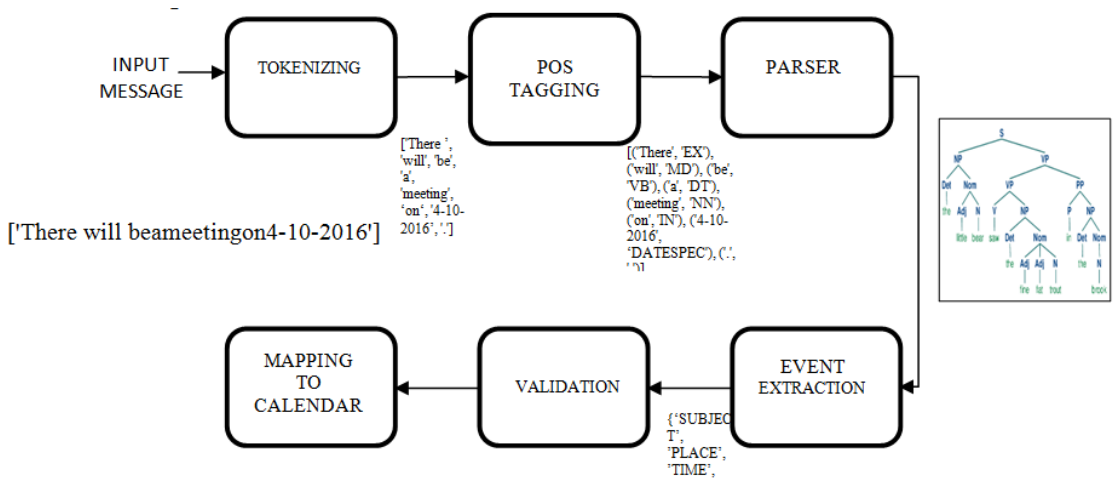

Fig 4: Project pipeline

3.1 Tokenizing

The message obtained by the user is stored as a string and it contains a group of words. The string is to be split into tokens. For doing this we use a splitstr() function. splitstr() function takes the string as the input and separates each words as tokens by checking the whitespace and periods. Each of the words are stored in an array. The array is returned by the function. These tokens thus obtained are passed to POS tagger. The tokens obtained contains words, brackets, '.'. Normal split() function separates the words based on white space only.
To optimize the processing, the string like"We cordially invite you", we hereby inform you", etc. are removed from the string and then tokenized.

3.2 POS Tagging

Noun, verb, propositions, etc. are the different part of speech tagging. Different part of speech tagging should be introduced in addition to the common POS tagging present in the English grammar for making date and time processing easy. For adding the additional POS tagging to the grammar we have defined a list of regular expression as rgt. The list of POS tagging is shown in the table.

TABLE I: POS Tagging

SAL	Salutation
EX	Expression
PUN	Punctuation
DATESPEC	Date specified
DATEUNS	Date unspecified
CON	Conjunction
DT	Determinant
VB	Verb
MD	Modal
DAYS	Day
MONTH	Month
TM	Time
NNS	Noun
IN	Proposition

The additional tags introduced in are DATESPEC for dates which are in the standard format, DATEUNS for the date which are not in the standard format, DAY for Monday, Sunday, etc. MONTHS for months (January, February, etc.), TM for time. Except of TM all the others are additional are used for making the date processing easy.

The regular expression define are passed to nltk.RegexpTagger() to convert regular expression to tags as regexp_tagger. Tokens obtained after tokenizing are passed to the regexp_tagger.tags which returns tokens with their POS (part of speech) tags. An example is shown below.

Text message:

```
there will be a meeting today.
```

After POS tagging:

```
[('there', 'EX'), ('will', 'VB'), ('be', 'VB'), ('a', 'DT'), ('meeting', 'NN'),
('today', 'DATESPEC'), ('.', 'PUN')]
```

3.3. Parser

Word which are obtained are after POS tagging may not be contain event information. These words are to be eliminated. Grammar is defined so as to eliminate the words which are of no use. There are two part in the grammar: chunking and chinking. Chunking are the words which are required for processing and chinking are the words which are not required for processing. Chunking part is defined in {} and chinking part is defined in } { . The grammar used in this project is shown below:

```
grammar = r"""

      NP: {(<SAL>|<DATESPEC>|<TM>|<NNS>|<IN>|<NNS>|<CHANGE>|<DAYS>|<PUN>|<DT>|<NN>|<CON>|<VB>|<DATEUNS>|<MONTH>)*}
      }<VBD|EX>+{

"""
```

Since verb and expression doesn't contain the information those are eliminated.

This grammar is passed to nltk.regexpparser() and the result is stored in cp. Words with their POS tags are passed to cp.parse(). Regexppraser is used to find the chunk structure for a given sentence, the RegexpParser chunker begins with a flat structure in which no tokens are chunked. The chunking rules are applied in turn, successively updating the chunk structure. Once all of the rules have been invoked, the resulting chunk structure is returned.

3.4. Event extraction

An event contains details like subject, place, date and time. A message containing event may or may not contain all the details. So the event extraction process is divided into 4 parts namely subject extraction, place extraction, date extraction and time extraction.

3.4.1. Subject extraction

Subject of the message is the reason for which the event is held. In most of the case subject appears to be the first noun. The subject can be multiword. So the word which are adjacent are also processed to check whether it is a part of the subject. If the adjacent words are noun, conjunctions or digits then they are added to subject.

3.4.2. Place extraction

Most of the messages the location of the event is specified comes after words like in, at, near, @,etc. These are stored in an array. The tokens are checked for these word and the noun appearing after these words are added to place. As is the case of subject, place can be of multiword. So the adjacent nouns appearing after the words specified are added to the place.

3.4.3. Date Extraction

The standard format used for date is DD/MM/YYYY. Date specified all the other forms are converted to this format. There are two ways to specify the date. One is in the direct form as shown above and some variants of it like DD/MM/YY, DD-MM-YYYY, DD-MM-YY,etc.

The other ways are like today, tomorrow, this Sunday, next Monday, 2nd of April, etc. for each of these case different method should be applied. For example for the case were "this Sunday" is specified we have to get value of the present day like for Sunday is 0, Monday is 1,so on. Then we have to get the difference in value of the day specified in the message and present day and add that difference to present date. For the case of "next Sunday" we have to do the same procedure and then add 7 to the date. For the case of "2nd April" we have to replace the date and month field of present date with the date specified in the message.

For performing the date operation we use date module in python. timedelta() function is used for the changing the date by specifying the number of days. Various other function are also used for the returning the month and day value.

3.4.4. Time Extraction

Time can also be specified in many ways like 930 am/pm,9.30 AM/PM,9, etc. time which have am or pm can be directly identified from the message and can be added to the time. But for the other cases the digits appearing after words like at, from, after, etc. are added to time. By default am is given to the time which doesn't have the am or pm part.

3.5 Validation

It is the process were the event details are checked for validity. If an event contain the subject and any other part then they are taken as a valid event. But there can be cases were the date and time extracted can be invalid. To check for date validity, date obtained are passed to check_date() function which return 1 if it is true and 0 if it is not. If date is invalid the date part is set are null.

3.6 Mapping to Calendar

The process of converting to an application involves in this stage. The extraction process is implemented with the python language and converting to application an integration process, to integrate java classes with python, is required as Android works well with java. The java platform can also be used for extraction since they are not much efficient compared to python this process is involved so as to have better results with efficiency. Inorder to convert the python library pyjnius is used which grants access to java classes. The reflection in java is made used here where it makes to inspect classes, interfaces, fields, and methods at runtime, without knowing the names of the classes, methods etc at compile time.

Classjnius. PythonJava Class: Base for creating a Java class from a Python class. This allows to implement java interface completely in Python. A Python class is created that mimic the list of declaredjavainterfaces. When an instance of this class to Java is given, Java will just accept it and call the interfaces methods as declared. Under the hood, we are catching the call, and redirecting to use the declared Python method. The class will act as a Proxy to the Java interfaces. Define at minimum the javainterfacesattribute, and declare java methods with the java_method() decorator.

*javacontext*Indicate which class loader to use: 'system' or 'app', default to 'system':

- By default, we assume that you are going to implement a Java interface declared in the Java API. It will use the 'system' class loader.
- On android, all the java interfaces that you ship within the APK are not accessible with the system class loader, but with the application thread class loader. So if you wish to implement a class from an interface you've done in your app, use 'app'.

jnius.java_method(java_signature, name=None)

Decoration function to use with PythonJavaClass. The *java_signature* must match the wanted signature of the interface. The *name* of the method will be the name of the Python method by default. You can still force it, in case of multiple signatures with the same Java method name.

Another library in use is the kivy, an open source library for developing mobile applications with a Natural User Interface (NUI). Kivy framework provides all elements required for producing an application such as:

- extensive input support for mouse, keyboard, TUIO, and OS-specific multitouch events,
- a graphic library using only OpenGL ES 2, and based on Vertex Buffer Object and shaders
- a wide range of Widgets that support multitouch
- an intermediate language (Kv) used to easily design custom Widgets.

In the upcoming android devices every element is important so that our application is compatible and get away from bugs. The kivy language is used to describe the user interface and interactions. Thus the access to java classes is made with the help of pyjnius and kivy and now a module which links the extracted date to the device calendar which is in java. The subject and other details extracted are added to the device. Since to make it general the whole process we bind all these using a tool called the buildozer which will automate the whole process and generates an apk. Buildozer will initialte downloads and sets up all the prequisites for python-for-android, including the android SDK and NDK, then builds an apk that can be automatically pushed to the device.

IV. Experiment Results

The major aim of the proposed work is to develop an application in order to automate the process of event extraction from text messages and add them on to the calendar. We considered a large set of messages and from each we extracted the event date place and time if its present in that.
Consider the following messages and the extraction process.

CASE 1: When all the details are specified:

a. The criteria 8 coordinators meeting will be at 4.10pm today in conference hall. Inconvenience is regretted.

[('The', 'NN'), ('criteria', 'NN'), ('8', 'TM'), ('coordinators', 'NN'), ('meeting', 'NN'), ('will', 'VB'), ('be', 'VB'), ('at', 'IN'), ('4.10pm', 'TM'), ('today', 'DATESPEC'), ('in', 'IN'), ('conference', 'NN'), ('hall', 'NN'), ('.', 'PUN'), ('Inconvenience', 'NN'), ('is', 'VB')]
(NP The/NN criteria/NN 8/TM coordinators/NN meeting/NN will/VB be/VB at/IN 4.10pm/TM today/DATESPEC in/IN conference/NN hall/NN ./PUN Inconvenience/NN is/VB)
Subject: The criteria 8 coordinators meeting
Place: conference hall
Date: 16-04-2016
Time: 4.10pm

b. Dear sir, a meeting of Research Council is scheduled at 3 PM to 3.20PM on 29th February in the Conference hall. All members are requested to attend..DrChithraprasad, Dean, Research

[('Dear', 'SAL'), ('sir', 'SAL'), (',', 'NN'), ('A', 'NN'), ('meeting', 'NN'), ('of', 'IN'), ('Research', 'NN'), ('Council', 'NN'), ('is', 'VB'), ('scheduled', 'VB'), ('at', 'IN'), ('3', 'TM'), ('PM', 'NN'), ('to', 'IN'), ('3.20PM', 'TM'), ('on', 'IN'), ('29th', 'DATEUNS'), ('February', 'MONTH'), ('in', 'IN'), ('the', 'DT'), ('', 'NN'), ('Conference', 'NN'), ('hall', 'NN'), ('.', 'PUN'), ('All', 'NN'), ('members', 'NN'), ('are', 'NN'), ('requested', 'VB'), ('to', 'IN'), ('attend..Dr', 'NN'), ('Chithraprasad', 'NN'), (',', 'NN'), ('Dean', 'NN'), (',', 'NN'), ('Research', 'NN')]
(NP,/NN A/NN meeting/NNof/INResearch/NN Council/NNis/VBscheduled/VBat/IN 3/TMPM/NNto/IN3.20PM/TMon/IN
29th/DATEUNSFebruary/MONTHin/INthe/DT/NNConference/NNhall/NN ./PUN All/NNmembers/NNare/NNrequested/VBto/INattend..Dr/NN Chithraprasad/NN ,/NN Dean/NN ,/NN Research/NN),
Subject: A meeting of Research Council
Place: Conference hall
Date: 29-02-2016
Time: 3

CASE 2: When some details are not present:

a. All Accenture selects please be present at the APJ hall by 3.45pm.

[('All', 'NN'), ('Accenture', 'NN'), ('selects', 'NN'), ('please', 'VB'), ('be', 'VB'), ('present', 'NN'), ('at', 'IN'), ('the', 'DT'), ('APJ', 'NN'), ('hall', 'NN'), ('by', 'IN'), ('3.45pm', 'TM'), ('.', 'PUN')]
(NP All/NNAccenture/NNselects/NN please/VBbe/VBpresent/NNat/INthe/DTAPJ/NN hall/NN by/IN3.45pm/TM./PUN)
Subject: All Accenture selects
Place: APJ hall
Time: 3.45pm

b.There will be a meeting at 4-10-2016 at apj hall.

[('There', 'EX'), ('will', 'VB'), ('be', 'VB'), ('a', 'DT'), ('meeting', 'NN'), ('on', 'IN'), ('4-10-2016', 'DATESPEC'), ('in', 'IN'), ('apj', 'NN'), ('hall', 'NN'))]
(NPwill/VBbe/VBa/DTmeeting/NNon/IN4-10-2016/DATESPEC in/INapj/NN hall/NN)
Subject: meeting
Place: apj hall
Date: 4-10-2016

Case 3: Personal Message:

a. lets hangout today .

[('lets', 'NNS'), ('hangout', 'NN'), ('today', 'DATESPEC'), ('.', 'PUN')]
(NP lets/NNS hangout/NN today/DATESPEC ./PUN)

Subject: hangout
Date: 16-04-2016

b. party at 4.
[('party', 'NN'), ('at', 'IN'), ('4', 'TM'), ('.', 'PUN')]
(NP party/NN at/IN 4/TM ./PUN)
Subject: party
Time: 4

Case 4: Message with incorrect output:
a. Are you free today.Letsmeeet at 4 pm .
[('Are', 'NN'), ('you', 'NN'), ('free', 'NN'), ('today', 'DATESPEC'), ('.', 'PUN'), ('Lets', 'NNS'), ('meeet', 'NN'), ('at', 'IN'), ('4', 'TM'), ('pm', 'NN'), ('.', 'PUN')]
(NPAre/NN you/NN free/NN today/DATESPEC ./PUN Lets/NNS meeet/NN at/IN 4/TM pm/NN ./PUN)
Subject: Are you free
Date:18-04-2016
Time:4 pm

b. My brother's engagement is on 4th April and wedding on 6th may. You and your family are invited.
[('My', 'NN'), (''brother's'', 'NN'), ('engagement', 'NN'), ('is', 'VB'), ('on', 'IN'), ('4th', 'DATEUNS'), ('April', 'MONTH'), ('and', 'CON'), ('wedding', 'NN'), ('on', 'IN'), ('6th', 'DATEUNS'), ('may', 'MONTH'), ('.', 'PUN'), ('You', 'NN'), ('and', 'CON'), ('your', 'NN'), ('family', 'NN'), ('are', 'NN'), ('invited.', 'IN')]
(NP My/NN brother's/NN engagement/NN is/VB on/IN 4th/DATEUNS April/MONTH and/CON wedding/NN on/IN 6th/DATEUNS may/MONTH ./PUN You/NN and/CON your/NN family/NN are/NN invited./IN)
Subject: My brother's engagement
Date: 4-04-2016

TABLE II: Test-case results

TYPES OF MESSAGE	ACCURACY
Message with complete information	48 out of 50
Message with complete information	44 out of 50
Informal messages	14 out of 20

V. Conclusion

Table II gives details of the results obtained from the test cases. Still this project came be developed with use of AI(Artificial Intelligence) by which we can train data set can obtain more accurate results. Messages can be discarded if it is not relevant. By using classifiers like Bayesian classifier, the messages can be classified initially so that we can avoid each message to be taken through all these processing steps. Further, this application can be implemented as a standalone. It can be extended to application which does not require network. As of now a server support is needed for the application to work, this can be made to a standalone one so that the application can work without a server after the application is installed. Also this project cannot produce mapping for message with multiple events.

References

[1]. Julie A. Black and Nisheeth Ranjan. Automated *Event Extraction from Email*. 2004.
[2]. Deyu Zhou,Liangyu Chen, Yulan He, " *A Simple Bayesian Modelling Approach to Event Extraction from Twitter*", *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, vol. 2, pp.700-705*, Jun 2014.
[3]. Cunningham, Hamish, et al. "Developing *Language Processing Components with GATE (a User Guide).*"2003.
[4]. Manning, Cristopher D. and Hinrich Schütze. *Foundations of Statistical Natural Language Processing. Cambridge, MA: The MIT Press*. 2003.
[5]. Califf, Mary Elaine and Raymond J. Mooney. "*Bottom-Up Relational Learning of Pattern Matching Rules for Information Extraction.*" 2003.
[6]. Choong-Nyoung Seon, Harksoo Kim and Jungyun Seo, "*A Efficient appointment information extraction from short messages in mobile devices with limited hardware resources, Pattern Recognition Letters," ACM Pattern Recognition letters, vol. 32, pp. 127-133*, Jan 2001.
[7]. Richard Cooper, Sinclair Manson, *Extracting information from short messages, Springer Berlin Heidelberg, 2007, vol. 4587, ch. Data Management. Data, Data Everywhere, pp. 224-234.*
[8]. Stuart Russell and Peter Norvig. *Artifical Intelligence: A Modern Approach. Second Edition. Pearson Education*, Inc. 2003.
[9]. Dalli, Angelo. "*Automated Email Integration with Personal Information Management Applications*." 2004.