Natural Language Processing

Minhal Khan

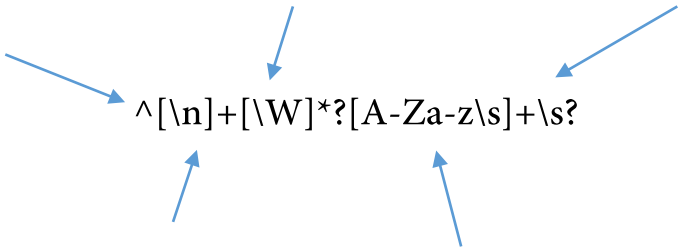
University of Birmingham

Tagging Seminar Information

Before we can tag, we need some form of data structure to store our email contents; to do this I had created a class 'EmailFile', which takes the argument 'number', this is the number of the *.txt* file within our test/training sets.
After loading an email we can see it has two parts to it, a header and a body; header contains routing information, such as sender, date, subject etc. whereas the body, is where the real meaning of an email is found.
Since the header is where information such as time, location and speaker could be found, requires us to split the email into two; header and body. And if we don't find information in the header we will try looking for it in the main body.

Splitting header and body:
Looking at several emails we can identify the phrase 'Abstract:' in-between the header and body and could run *.split('Abstract:'* however, not all emails do therefore we will build a **regular expression** which will encapsulate a similar split.

This expression is:



Now we have the header and body separately, we can start tagging!
We will begin tagging by tagging time of the lecture due to its regular structure:

As we can see, two numbers are separated by a semicolon and (an optional) some time may include the ante/post midday. Further, we can't take any two numbers

separated by a semicolon therefore we will need to restrict its range where the hour range being 0-24 and the minutes 0-59.

hour range = ([0-9]|0[0-9]|1[0-9]|2[0-3]) (We also include 0[0-9] as we represent hours using double digits by padding it with a 0 e.g. 09 instead of 9)

minutes range = ([0-5][0-9]) (minimum minutes = 00, maximum minutes = 58)

optional AM PM = [\s*]([AaPp][\.]?[Mm])?[^$]? (We add a '\s*' as it may be right after the minutes or a couple of spaces from it, this will help captures these unknowns)

Joining them together we get this regex (**let's call this singleTime**):

'([0-9]|0[0-9]|1[0-9]|2[0-3]): ([0-5][0-9]) [\s*]([AaPp][\.]?[Mm])?[^$]?'

If we start tagging using this expression, we may end up tagging a time which is the end and not <*stime*> of the lecture. To prevent this, we will create another regex to capture the structure of two times; '7:00 8:00' or '7:00 to 8:00' and run this regex first, therefore if we try tagging something that's already tagged our tagger will stop this from happening.

(singleTime [-\s*?to\s*?] singleTime)

Finally, a time can be written in a small format; 'Hour AM/PM', again we will build a regex to tag this:

([1-9]|1[0-2])[\s*]?([AaPp][Mm])

Tagging Paragraph/Sentences

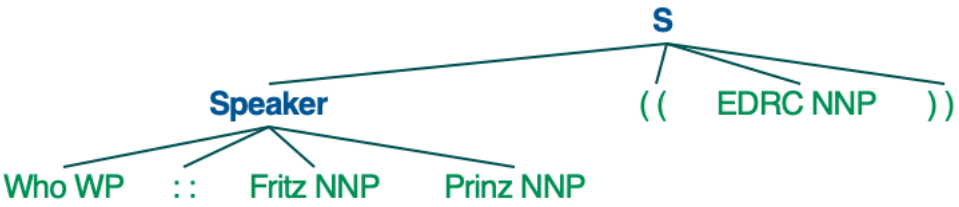
To tag sentences I will simply use NLTKSs' method '.sent_tokenize' to split up the sentences on the body of the email, to prevent us tagging the header unnecessarily.

To tag paragraphs, I loop through each line and check if there is a line split if so it is the end of the paragraph

Tagging Speakers

To start tagging speakers, we will try and find the speaker within the header.
It will usually be found within the 'Who:' tag. Since we have already used regex we will build a POS Tree and pick out the 'Who:' contents; done by chunking the tree and taking anything within a (Who) WP tag or (Speaker) NN, <:> and NNP (to get names)

Who
Chunking expression = "Speaker: {<WP><:><JJ|NN|NNP>+<NNP+|\.|FW>*}"



As we can see, it identified the speaker, however what if it has other entity tags e.g. University?

As we can see, it is setting university as a child node to Speaker, this is clearly wrong. To remove such issues, I will be using StanfordNERTagger to tag the sub-tree, identifying each entity.
Running the tag above will return:

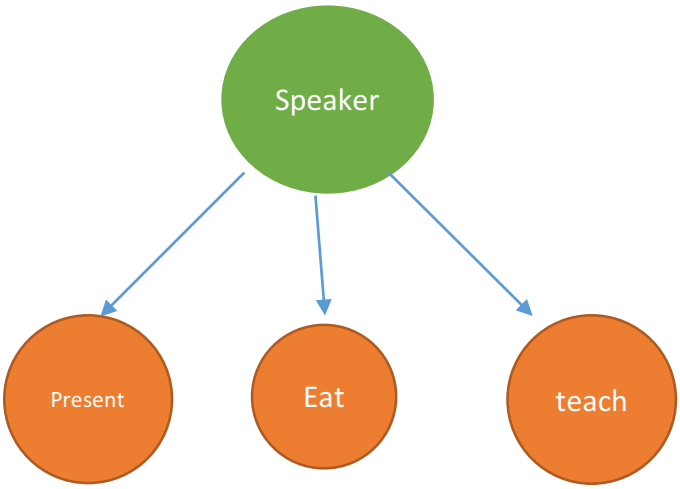
[(Gian, PERSON),(Aldo, PERSON),(Antonelli, PERSON), (Univ, ORGANISATION)]

Using this tagged array, we can easily pick out the speaker!

What if we still don't find the speaker?
We will try running the same process on the body of the email. However, some emails have the speakers name within the main text of the email (not under a tag). To extract this, I have built a **Naïve Bayes classifier** (speaker or not speaker; binary output). (Since the Stanford tagger can identify people); given a list of names, how do we know it is the speaker?

Answer; verbs used to describe the speaker are usually different to verbs used in non-speaker sentences. (Pr(speaker | [verbs]))

Building the classifier
(Visual representation)



Its more probable that the word *present* will occur within a sentence the speaker is in, whereas it is less probable the speaker will be in a sentence with the verb 'eat'.

Steps in building the classifier (see *Bayes.py*):

1. Load the training emails
2. Get the speakers name by looking at the speaker tags (done using a regex <speaker>(.*?)<\/speaker>)
3. Run StanfordNERTagger to get all names within the email
4. The Stanford tagger will pick up the speaker's name, therefore we will remove it from the list of names thereby only giving names that is not the speaker
5. Get all of the sentences in which the speaker is in
6. For each sentence get all the verbs (I done this using Chunking with the expression 'Verb: {<VB.?>*}')
7. I will then lemmatize the verbs (e.g. presenting --> present) Once lemmatized, we will push this word as a tuple with an outcome of True (since it appeared in a sentence where there is a speaker) onto our training set
8. We do the same but on the list of names we found that are not the speaker (and put the verbs with the outcome being False)
9. We do this for every file within the training folder (0-300) and store the verb array in a .txt file (called 'Bayes.txt')
10. At this point we have built our training data for our classifier, we just have to feed this into nltk.NaiveBayesClassifier and store the classifier (called bayes_classifier.pickle')

Once we do this, we can classify if a list of verbs in a sentence is a speaker or not:

```
>>> x = NaiveBayes()
>>> print(nltk.classify.accuracy(x.classifier, x.feature_set[600:]))
0.7209302325581395
```

```
>>> x = NaiveBayes()
>>> x.isSpeaker("present")
test_sent: present
tag: True
>>> x.isSpeaker("eat")
test_sent: eat
tag: False
>>> x.isSpeaker("teach")
test_sent: teach
tag: True
>>> x.isSpeaker("give")
test_sent: give
tag: True
>>>
```

As we can see the assumptions we had made at the start can be clearly seen as it returns True for present, teach and give but False for eat.

Finally, given this classifier we can parse verbs (verbs will be lemmatized) from sentences in which the speaker is unknown and pass this into our classifier to decide if the name(s) in the sentence is a speaker. (Some sentences may have more than one verb we can still classify this by passing it as a single string)

```
>>> x = NaiveBayes()
>>> x.isSpeaker("give present")
'True'
```

Test Run of our classifier on (test_untagged/456.txt)

In this file there is no tag (e.g. Who, Speaker) for us to identify if it is the speaker as well as there are 3 names (two are the speakers and one is the sender), therefore will be using this to test the outcome of my classifier.

The Stanford Tagger clearly picks all 3 names (One name is the speaker but just its first name):

```
[' Catherine Copetas', ' Catherine Copetas', ' Kai-Fu Lee', ' Rick Shriner', ' Rick']
```

It should run the classifier on every sentence in which the name occurs and decide if that name is the speaker:

```
Kai-Fu Lee is a speaker
Rick Shriner is a speaker
Rick is a speaker
```

As we can see, it successfully was able to remove ‘Catherine Copetas’ who was the sender and even tag the abbreviated name ‘Rick’ by looking at the verbs in the sentence!

```
<speaker>Rick</speaker>
<speaker>Kai-Fu Lee</speaker>
<speaker>Rick Shriner</speaker>
```

Location Tagger

To tag location, I use the training data to gather all the locations (LocationTagger.py) and store them into a .txt file which will be used to identify locations within an email.

Evaluation

(Evaluate.py) To calculate **precision**, **recall** and **f-measure** I need to calculate tp, fp, tn, fn:

1. True Positive This is a tag my system was able to tag and in the test tagged file tag (line 55 in Evaluate.py)
   (Contents within my system tag match the contents within the test data)
2. False Positive This is a tag my system missed (tag contents within the test data is not in my systems tag content (line 57))
3. False Negative This is a tag that should not be tagged and my system tagged it (Tag contents within my system that does not exist in the test data tag (line 61)
4. True Negative - This is a tag that should not be tagged and my system did not tag it (Intuitively this would always remain zero for this system)

To check these, I get all the information within a tag using the regex '<TAG>(.*?)</TAG>' (this works well for all tags surrounded by text; speaker, location, s/etime) however running this on sentences/paragraphs will miss those that start or have a new line within it therefore I use this expression to get the contents within a paragraph and sentences; ([\W]*?.*?\s.*?\n.*?\W.*?)

Ontology Construction

To create our topic structure I have created a method, which extracts the topic (regex; (Topic: \s.*.*?)) and counts the number of occurrences of a word.

<- This is what my algorithm extracted

And using this I manually constructed an ontology tree:

→ ai
→ mathematical
→ graphics

→ robotics
→ vision
→ logic
→ simulation
→ imagery

→ computer science
→ system
→ programming

→hci

In order to tag each email to its corresponding topics I calculate semantic difference, (steps in calculating this):

1. I load the email file and tokenize it using WordPunctTokenizer to help split the email into sequence of alphabetic and non-alphabetic characters
2. We then try matching as many pre-defined categories to this email
3. We then loop through every valid word (.isalpha) and locate the lowest single hypernym that is shared between the category and word
4. Finally, we loop through elements generated at steps 2 and 3 and calculate the semantic difference. To do this, we use wordnets .path_similarity to path a similarity to the computer science synset (as the emails are all on computer science related topics in some way) and pick the one with the lowest semantic difference (line 114)

Once we do this, we can determine the topic of a email and store them in there correct directories.