**Created by: Saurabh Jain**

**On: : 04/07/2016**

This file contains information about how to run this project and share the details of algorithm used in this project.

**List of libraries used:**

• PyEnchant lib to get words in English vocabulary.

• metaphone module written by Lawrence Philips for Double Metaphone Algorithm

• Nltk to get access to the tokenizer and corpus

**List of corpus used:**

• Gutenberg corpus is used to compute phonemes of English words

• Brown corpus of categories news, editorials and reviews is used to compute unigram, bigram and trigram probabilities of English words.

**Project structure:**

ruleslangdetection.py : contains rule based logic for slang words detection

mlslangdetection : contains machine learning based logic for slang word detection

metaphone.py : contains logic for double metaphone algorithm

main.py : main file, starting point of the project

data/

unigram : unigram probabilities of English words, separated by ‘@$@$’

bigram : bigram probabilities of English words, separated by ‘@$@$’

trigram : trigram probabilities of English words, separated by ‘@$@$’

domain\_names : list of registered domain names

slang dict.csv : list of slang word to corresponding formal English word

testdata : test data containing text with slang words

words\_to\_metaphone.csv : list of mapping from word to its meta phonemes

**How to run this project:**

• Change directory to the root directory of project.

• Run following command: **python main.py**

• Output will be in file ‘**data/output**’. Paragraphs will be separated by ‘\n@@@@@’ in output file.

**Algorithm Used in Project:**

Step 1) Create object of **RuleSlangDetection** and **MLSlangDetection** class and Initiate all data

structures required in this project. Like unigram\_probs, bigram\_probs, trigram\_probs,

metaphone\_to\_words, slang\_dict, domain\_names.

Step 2) Read one paragraph from **‘testdata**’ file and call function ‘**parseParagraph**’.

Step 3) **‘parseParagraph**’ will tokenize string into words token using ‘**WhitespaceTokenizer**’ and save all tokens in a list ‘**words**’

Step 4) for each ‘**word**’in list ‘**words**’ if (‘**word**’ has length 1 or ‘**word**’ is not in English

vocabulary):

[\*\*used PyEnchant English dict to check In Vocabulary word and pass this check for word having length 1 b/c of the words like ‘**u**’ which are in vocabulary but still is a slang word\*\*]

a) Check whether ‘**word**’ is a HTML entity. Like ‘**&amp;**’ should be replaced with ‘**&**’. I used HTMLParser module for performing this task.

b) If ‘a’ step is not true then check whether ‘**word**’ contains punctuations due to which vocabulary check has been failed like ‘**thanks,**’

c) If ‘b’ step is not true then check whether ‘**word**’ is a number like “**-32.09%**”

d) If ‘c’ check is not true then check whether there are multiple words separated by delimiters due to which In Vocabulary check is failing like ‘**install/reinstall**’

e) If ‘d’ step is not true then check whether the given word is in ‘**slangDict**’

f) If all of the above cases got failed, then machine learning approach is used to convert slang word into correct formal English word.

[\*\***DESCRIPTION OF MACHINE LEARNING APPROACH**

Let’s suppose we have slang word **‘W**’ and its correct English formal word is **‘C**’. Then we need to compute the probability that the correct word is **‘C**’ given incorrect/slang word **‘W**’ i.e **P(C/W),** probability of **‘C**’ given **‘W**’**.**

Now using Bayes theorem:  **P(C/W) = ( P(C) \* P(W/C) ) / P(W)**

Now P(W) will be same, so let’s focus on **P(C) \* P(W/C).** Now for all candidate words ‘**C**’ which can be a correct formal english word of slang word ‘**W**’ we will pick the word for which **f(C) = P(C) \* P(W/C)** is maximum. Now **P(C)** is simply a language model which tell, in a text what is the probability that next word will be ‘**C**’. So using a training corpus and statistics we can compute **P(C).**

Now **P(W/C)** can be computed from a error model. It’s simply, what is probability of getting incorrect word ‘**W**’ if correct word is ‘**C**’.

**Candidate words:**

Now next challenge is to get candidates word which might be correct formal word for slang word ‘**W**’. I have considered three types of words as candidate word :

Type 1 : words which are One Levenshtein distance from the given slang word ‘**W**’

Type 2 : words which are Two Levenshtein distance from the given slang word ‘**W**’.

Type 3: words having phonemic which is One Levenshtein distance from the phonemic

of the given slang word ‘**W**’.

Then I check each candidate word whether it is in vocabulary or not and remove out of vocabulary candidate word. After that I compute **P(C)\*P(W/C)**  for each candidate word and word which has maximum value will be considered as the correct formal word of the given slang word.

**NOTE:** As I could not implement error model, so I used constant error model probability (**P(W/C)**) in this project but different for Type 1, Type 2 and Type 3 candidate words as:

**ERROR\_MODEL\_PROB\_ONE\_EDIT\_DISTANCE = 0.6**

**ERROR\_MODEL\_PROB\_TWO\_EDIT\_DISTANCE = 0.4**

**ERROR\_MODEL\_PROB\_PHONEMIC\_ONE\_DISTANCE = 0.6**

\*\*]

Step 6) If all the above checks got failed, then we use machine learning technique to compute

correct formal word. We pass current word ‘**word**’ and two previous word of ‘word’,

‘**prev\_one**’ and ‘**prev\_two**’ .

Step 7) Now we parse the three words ‘**word**’, ‘**prev\_one**’ and ‘**prev\_two**’ to get correct words

for statistical processing. For ex, if the three tokens are ‘.**This**’, ‘**tuesday**’,

‘**yesterday**’ then correct three words will be

current\_word = **‘This’**

prev\_one = ‘**.**’ [‘**.**’ + ‘**This**’]

prev\_two = **‘tuesday**’

Step 8) Now after parsing the words, we calculate candidate words for ‘**current\_word**’ i.e words

which are One and Two Levenshtein distance from ‘**current\_word**’ and words having

phonemic One Levenshtein distance from phonemic of ‘**current\_word**’. We used

‘**metaphone\_to\_word**’ dict to calculate ‘**Type 3**’ candidate word.

Step 9) Compute the value of **P(C)\*P(W/C)** for each candidate word. I used following formula to

compute **P(C):**

**trigram\_prob(current\_word, prev\_one, prev\_two) \* 0.7 +**

**bigram\_prob(current\_word, prev\_one, prev\_two) \* 0.3**

Step 10) Return the word for which **P(C)\*P(W/C)** is maximum as the correct english formal word for given slang word ‘**W**’.

Step 11) After performing all steps for each word ‘**word**’ in list ‘**words**’, convert the paragraph in

sentences using ‘**nltk.sent\_tokenize’** and write output in file **‘data/output**’. All

paragraphs will be separated by ‘**\n@@@@@’**.

**NOTE:**

1)I analysed my algorithm, it is correct but not giving efficient results like for word ‘**rplace**’ it is outputting ‘**place**’ as correct word not ‘**replace**’, although it has ‘**replace**’ in candidate set. So I analysed it for lots of example, I got to know that problem is with dataset which I am using at training time to compute **trigram, bigram and unigram** probabilities. For ex, for word sequence “**I want replace**”, there is no **trigram and bigram** corresponding to this sequence in training dataset, so **trigram\_prob and bigram\_prob** returns zero.

So one solution to this problem might be **Smoothing,** handle cases which are not occurring in training set.

Other solution is to use the right ‘**Corpus**’ for training. I was not able to find **chat corpus with formal english words**, so I trained it on **news, editorial and review** categories of Brown corpus.

2) I was getting weird results by using **unigram\_prob** because, one, my training data set is not good enough and second, unigram\_prob uses the probability of occurrence of a word independent of any other word, so later I commented code belonging to the **unigram\_prob.**

3) If my training dataset would have been correct and I would have used the correct **Smoothing** techniques then I have added the ML output **slang\_word -> correct\_formal\_word** matching in **slang\_word dict,** so that if next time that word occur in the text, we do not need to run ML algorithm again for already parsed slang word.