

CS6370:NLP Assignment 1 – Spell Check

-Submitted By:

Priya Prakash (CS10B047)

Shivam Srivastava (CS10B051)

Utkarsh Srivastava (CS10B058)

Introduction

Assignment statement: Spell Check for - standalone words, phrase check and errors in sentences.

Requirements: NLTK Corpus (Brown Corpus, WordNet) ; N-Gram Models

Programming Language Used – Python

Output Format – Test cases are checked based on Mean Reciprocal Rank measure.

Standalone Word Spell Check

Stand-alone words are given (correct/incorrect). Identify the best possible correction based on a set of parameters obtained from the corpus used.

Basic Idea: a) Single letter substitution/deletion/insertion/transposition are the more probable errors found while word spell checking.

b) Hence, initially a bag of words containing all the candidate/possible words is constructed by applying the single letter subs/ins/del/trans on the input word.

c) To go a level deeper, a bag of words_2 is constructed from all the candidates (exist/don't exist in the dictionary) constructed in level 1.

d) Finally, the meaningful words out of all these candidates are filtered out and later ranked based on certain defining parameters.

e) The above computation takes $O(n^2)$ time, where n is the length of the input word, such that:

Level 1: Insertions: $26(n+1)$ Deletions: n
Substitutions: $26n$ Transpositions: $n-1$
= $(54n+25)$ computations

Level 2: $(54n+25)$ computations for each candidate at level 1.

Hence, Total Time taken is approximately $\sim O(n^2)$.

[Limitation: larger the word size, more is the time needed for computation.]

f) Once the filtered words are generated, the Levenstein-Damerau distance is used to find the distance to which a word can be changed to another word. Here the possible candidates are matched with the erroneous input word. Different weights are given to ins/del/trans/subs based on the probability of their occurrence based on their general usage.
(Here: ins: 1.25; del: 1.5; subs: 1 ;trans: 1)

g) Now 4 Parameters define the final rank order of the possible word candidates:

- i) **Measure 1:** inverse of the edit distance obtained. Closer the word to the input erroneous word, lesser is the edit distance computed.
- ii) **Measure 2:** Single letter errors are given higher weightage as they are more probable in practice. (~ 0.9995 v/s 0.0005 for 2-letter error change)
- iii) **Measure 3:** Probability of the occurrence of the word in the Corpora. If the word info is not found, we do appropriate smoothing.
- iv) **Measure 4:** Keyword Confusion Matrix for ins/subs. This is based on the fact that keys that are spaced close by on the keyboard have a higher chance of getting subs/ins instead of letters farther on the keyboard.

h) In the end, a combined measure = $M1 * M2 * M3 * M4$ is computed which finally ranks (descending value) in order, the possible corrections and displays the top 7 results.

Implementation Details:

- a) How and why did we compute the Confusion Matrix (Keyboard Spacing)?:

This Matrix ($26*26$ – for every alphabet) is only defined for subs/ins; but at the same time, there is no other measure to balance out this measure multiplication for del/trans.

Hence, to balance and not bring about biased changes, the values in the matrix are very close to 1. For instance, $M[a][s] = 0.9999$ and $M[a][d] = 0.9998$. The value is assigned based on the distance of the letters from each other on the qwerty keyboard.

So, even if no extra measure for del/trans is used, a small change in 0.0001 will not affect the ranking amongst words made from del/ins/subs/trans but re-arrange words formed using ins/subs amongst themselves based on this extra measure.

(eg: error word: chim – possible candidate : chin/chip ; here chin is more probable than chip (not taking any other measure into account as of now) because 'n' is closer to 'm' than 'p' on the keyboard.)

```

Enter Word:chim
#The edit distance is based on the DP
Input word: chim
def weight_calc():
    Level 1 words generated
    Level 2 words generated
    words filtered
    Edit distance done
    Weights calculated
    for b in filteredWords:
    Printing top 6 results with the weights:
    him 0.00204759911423
    chin 1.59445866876e-05
    chip 1.09401747215e-05
    chic 4.68817730388e-06
    whim 1.56288210307e-06
    his 1.21593975098e-06
  
```

```

Enter Word:srim
Input word: srim
Level 1 words generated
Level 2 words generated
words filtered
Edit distance done
Weights calculated
Printing top 6 results with the weights:
trim 1.28156332452e-05
slim 1.28117874015e-05
krim 1.09357973384e-05
swim 9.69083831673e-06
grim 9.06471619781e-06
rim 4.29921554811e-06
  
```

Sample Example: chim; srim

b) How Is Edit Distance Computed? :

Our algorithm implementation is based on the Damerau-Levenshtein distance which also takes transposition into account other than ins/subs/del. Appropriate weights for these operations have been chosen based on general error pattern as mentioned earlier.

c) How did we find the value for Measure 3: Frequency of word in corpora?

It is assumed that possible candidate words that occur more frequently in the corpora than some other candidate words have a higher probability of being the correct/intended input word.

For the same, the count of each possible word is found in the corpora and divided by the No. of total Words.

But there may be a case where that word is present in the dictionary but not in the Brown Corpora info. For that smoothing needs to be done, as Laplace Smoothing:

$$(\text{Freq}(w) + 0.5) / (N + 0.5 * V)$$

Hence for words that have no freq count or are unseen with respect to our current corpora, get a very minimal value assigned to them which is not 0. At a later stage, this forms one of the crucial deciding factors to find the rank order.

Phrase/Sentence Spell Check

Phrases/Sentences are given (with a set of correct/incorrect words). Identify the best possible correction based on a set of parameters obtained from the corpus used.

Now the errors can be of the varied types (which the program can handle):

- a) Single word errors;
- b) Double word errors;
- c) Single Homophone errors;
- d) Splitting of a long word into its meaningful constituents;
- e) Combining a smaller set of constituents to a bigger meaningful word(s);
- f) No error – Completely Correct Sentence.

- **Single Word Error handling:**

Basic Idea & Implementation Details:

[Initially All the Ngram files are read and stored in a dictionary mapping format ; so is the brown corpus which is then broken up into bigram-trigram-

4gram model and also mapped to the same set of dictionaries to increase the info]

- a) The Sentence is parsed and is refined (removal of unwanted special characters from the sentence word set and change every word to lower case for uniformity (done while reading the dictionary as well)).
- b) Once this is done, the index of the erroneous word is noted and the possible candidates of the incorrect word generated using the above Measure/Results obtained from a normal standalone Word Spell Check. Use the **Rank order probability** obtained from here as **Measure#1** for this part.
- c) If no erroneous words are found send this to homophone error check; else if there is an error, then we anyhow get the possible candidates.
- d) If the word is in error but no possible words exist, try for splitting word error check.
- e) Else if none of the above conditions hold, call the specific function to handle Single Word Error.
- f) As the program enters “**getContextOfEachWord()**”, the **Measure#2** for the same is calculated which is based on **Language Models**. Given we have the error word at index ‘k’, we search through the corpora space to find similar matching occurrences of unigrams/ bigrams/ trigrams/ four-grams and get their respective counts to decide a rank order from this measure.
- g) Given error at index ‘k’ we search and relate to the context with respect to the neighboring words of the language model within a window size (say here 3) and get a cumulative result by combining the probabilities of all (multiplication) to get a descending rank order.

Now the ‘history’ size plays a very strong role in determining so. We take into account all sized history (2-3-4 as we have these kinds of Ngrams available) in which we find the word at index ‘k’ to get as much info about the word as possible in trying to determine the best possible fit out of all possible candidates.

Hence a general Measure#2 looks as follows:

Say the sentence looks like: (where all the Ngram info available from the dictionary mapping of each kind of dictionary)

S-n S-(n-1).....S-3 S-2 S-1 'K' S1 S2 S3.....S(n-1) Sn

Individual components:

- a) $P(K|S-1) = C(S-1 K)/C(S-1)$ (bigram/unigram)
- b) $P(K|S-2 S-1) = C(S-2 S-1 K)/C(S-2 S-1)$ (trigram/bigram)
- c) $P(K|S-3 S-2 S-1) = C(S-3 S-2 S-1 K)/C(S-3 S-2 S-1)$ (4gram/trigram)
- d) $P(S1|K) = C(K S1)/C(K)$ (bigram/unigram)
- e) $P(S1|S-1 K) = C(S-1 K S1)/C(S-1 K)$ (trigram/bigram)
- f) $P(S1|S-2 S-1 K) = C(S-2 S-1 K S1)/C(S-2 S-1 K)$ (4gram/trigram)
- g) $P(S2|K S1) = C(K S1 S2)/C(K S1)$ (trigram/bigram)
- h) $P(S2|S-1 K S1) = C(S-1 K S1 S2)/C(S-1 K S1)$ (4-gram/trigram)
- i) $P(S3|K S1 S2) = C(K S1 S2 S3)/C(K S1 S2)$ (4gram/trigram)

Based on which all probabilities are available for a given sentence, those are calculated based on the information from the dictionary and then multiplied.

Now there may be 3 cases while calculating these values:

All values from **a)-i)** probabilities are of the form **Num/Den** where Den>Num always.

Case 1: Num!=0 and Den!=0 : Substantial values of both found in the corpora and they can be divided to give some value having an appropriate weight.

Case 2: Num==0 and Den==0 : Both values don't exist, smoothened to a very low value of $10e-15$.

Case 3: Num==0 but Den!=0 : LaPlace Smoothing done. Here this probability term is parsed as:

$0.25/(den+0.25*size_of_vocab_from_dict)$

such that if values have some data,i.e.=Num!=0 then they are not touched but if no occurrence is seen in such case as above, we smooth the unseen values and give them a minimal weight.

Reason for such smoothing: Say we have a case bigram/unigram : such that Den!=0 but Num==0 ; Here let the words be: A = C(the tree)/C(the) v/s B = C(fire arm)/C(fire) with C(the)>C(fire) ; now the values are to be smoothened so that in the end A < B still.

Now we argue the following by saying that as C(the)>C(fire) there were more chances of seeing 'the tree' because there were larger no. of combinations available with us and we still did not see it, but at the same time there were low no. of combinations available and had

been there some more combinations available, we could have got 'fire arm' as one of the possibilities as well. Hence even after smoothing $A < B$ which is justified here by Laplace smoothing.

- h) Now the final **Measure#2** is calculated. But there may be some peculiar cases which do not give correct ordering. Example : due to unavailability of sufficient data from the corpora (insufficient knowledge) or high occurrence of a non-stop word which though is not the correct replacement, may get up high in the rank order due to a high unigram probability (eg-and, to, in, the etc.) For the same, two more measure are included to levy or balance up with this incapability as much as possible.
- i) **Measure#3** is to determine the **semantic relatedness** between the possible candidate and all the non-stop words in the sentence (eg: the house is made from 'brick' :brick corrected: SemRel(house,brick) and SemRel(made,brick)). For the same Wordnet is used and its associated path similarity/edge count measure.
- j) But due to a limitation of the wordnet, sometimes the wordnet similarity measure doesn't give the expected output value. Hence, we can't completely rely on SemRel as well. Hence we need to come to a balance point which gives equal weightage to the corpora ngram knowledge and SemRel and gives the best possible output. (Note-SemRel is not defined for stop-words). To Balance these things out, an extra **Measure#4** is used that tries to enhance the values received from Measure#2.
- k) Before Measure#3, **Measure#4** is calculated to save ourselves from some computations as follows: Now as the rank order is received from Measure#2, as discussed above, we need to balance things out between values from Measure#2 and Measure#3. For that we introduce Measure#4 which in turn segregates the rank order farther apart to enhance the correctness of data received from the Corpora.
- l) **Measure#4 Calculation:** We extract the power from the rank order and then find the mean and standard deviation. After that starting from the mean, we start making sub-ranges of the size: ***mean +/- (0.25*k)deviation***. Now higher the rank order (lower the absolute power) and hence it will lie in a much lower sub range. Say for example in rank order : -6>-17 and here in the range order 6 lies in [5.48,7.9] and 17 lies in [16.69-18.5].

Hence Measure#4 = $(1/(\text{start_value_of_the_sub_Range_it_lies_in}))^{**5}$.

m) Hence after applying this, each value gets degraded (due to multiplication with value <1). But the extent of degradation varies, which is defined here as 'magnification'. Now when a no. lesser than 1 is raised by power 5, it gets much much lower.

So, if $a < b$ and both $a < 1$ and $b < 1$, then $a^5 < b^5$. Hence due to the same fact, the magnification is done. If a no lies in a lower valued sub-range, means it is higher in the rank order (negative power) and hence magnified (rather de-magnified by a smaller extent due to smaller value of start range) and similarly if a no. lies in the higher sub-range, it gets multiplied with a much lesser value when raised to a power of 5.

n) Now, what happens if a stop word gets way high up in the rank? : It is generally seen from the test cases that if a non stop-word is a valid candidate for replacement then the corpora knowledge is sufficient for it to be on the top. And we have assumed that if a stop word is at the top, it is there not because of its high probability of occurrence but due to it being actually a much more probable candidate.

To ensure this aspect, we check if any non stop-word lies within a range of values from its bound (say $\pm 0.75 \times \text{deviation}$) . If no, we declare the final answer of replacement to be the stop-word and don't have to measure Measure#4 ; else if a non-stop word is found, within the range defined, we assume that the non-stop word was more probable but lay behind in the rank just by a few point tally.

Another case is when the top ranker is a non-stop word. In 2 such cases, the values are magnified and semantic measure is checked for all the probable non stop-words. As we have already figured out that stop word would be at the top rank if they are actually the replacement, in these 2 cases, we conclude that stop words are not so probable and move them down the line after calculating semantic measure. If they still have the weight due to some measure (not Measure#3) they will find their rank somewhere in the order.

o) **How is Measure#3 calculated?** : All the stop words are removed from the current sentence in context. The remaining word set is extracted out which does not contain the word that needs to be corrected. Now each word from these non stop-word set (W) is checked against the corrected word (C) for similarity using the Wu-Palmer similarity measure from WordNet (based on the distance from the Least Common Ancestor).

Now to check each 'W' set value against 'C' ; all probable synsets of 'W' and 'C' are calculated and the highest value taken. 0 values are handled separately, i.e. whose info cannot be found from the wordnet anyhow. For such cases, a transitive property is used. **How?**

We have 2 2-D arrays:

- i) First 2-D array contains the SemRel between all the non stop-words. ($W \times W$)
 - ii) Second 2-D array contains our final answer. Now the values are filled after finding SemRel between each 'C' (row) and the corresponding 'W' (column) set words. ($C \times W$)
 - iii) If some value is 0 in the second array (A (in C) and X (in W)). We look at all the non-zero entries in that row. Such that, say if a non-zero value A (in C) exists for some column B (in W) and B and X have some non-zero value in the First 2-D array. Find the max across such possible B -> say K.
 - iv) If still the value remains 0. No similarity could be found – enter a small minimal value: $10e-6$.
 - v) Else, we find the least possible entry in the respective column and lower down the value obtained as K to the $0.9 \times \text{minimum}$ in that column. Because anyhow it was initially 0 so it cannot exceed any value which was already found using the measure.
 - vi) From this, all the values in the SemRel matrix get filled.
 - vii) SemRel for any word in a sentence is nothing but the multiplication across all columns for that word. Higher the value, better is the similarity.
- p) Now that All the **Measure#1,2,3,4** are ready. We multiply all the terms and order the final result in descending order and check with the expected output. Mean Reciprocal Measure tells us how close is our answer to the expected result.
- q) Major computation time is required for reading the dictionary (done once) and computation of similarity measure for pair wise words.

```

Brown Sentences Read
2Gram Model Ready
3Gram Model Ready
4Gram Model Ready
2-3-4Gram Model from Brown Corpus Ready
Dictionary read
Enter sentence: From the eath to the moon.
Word = from
Input word: from
Correct word
Word = the
Input word: the
Correct word
Word = eath
Input word: eath
Level 1 bag of words generated for : eath
Level 2 bag of words generated for : eath
Possible Words Filtered Out
Edit distance Calculated for the current probable candidate
Weights calculated for Each Candidate
Printing top 7 results with the weights(Context Not Applied on Rank):
each 0.000548571618178
death 0.000216914966291
earth 9.41137367259e-05
eat 3.20486977223e-05
path 2.78053876098e-05
bath 1.65648931404e-05
oath 4.06186758283e-06
Possible Corrections of eath : ['each', 'death', 'earth', 'eat', 'path', 'bath', 'oath']
Word = to
Input word: to
Correct word

```

```

Input word: moon
Correct word x
Measures Being Evaluated
Non stop words in the sentence include: ['moon']
Before Magnification
earth 1.67967316573e-19
path 3.4362782523e-21
death 4.17055363782e-28
bath 2.67088765765e-42
oath 6.99370503357e-53
each 5.74088306948e-56
eat 1.93476219043e-68
Relative Magnification Begins (if applicable)
After Magnification!
earth 4.03870190095e-27
path 8.26238329746e-29
death 9.94946846238e-37
bath 6.07785586956e-52
oath 4.9218643869e-63
each 2.42464277113e-66
eat 2.196959107e-79
Rank After Adding All Valid Measures
earth 1.10957592041e-31
path 2.58060564843e-34
death 1.38364279309e-41
bath 3.99800006118e-58
oath 1.81458131428e-70
each 6.72763734547e-90
eat 1.01772949302e-96

```

Sample Example: “From the **eath** to the moon”

- Double Word Error handling:

Basic Idea & Implementation Details:

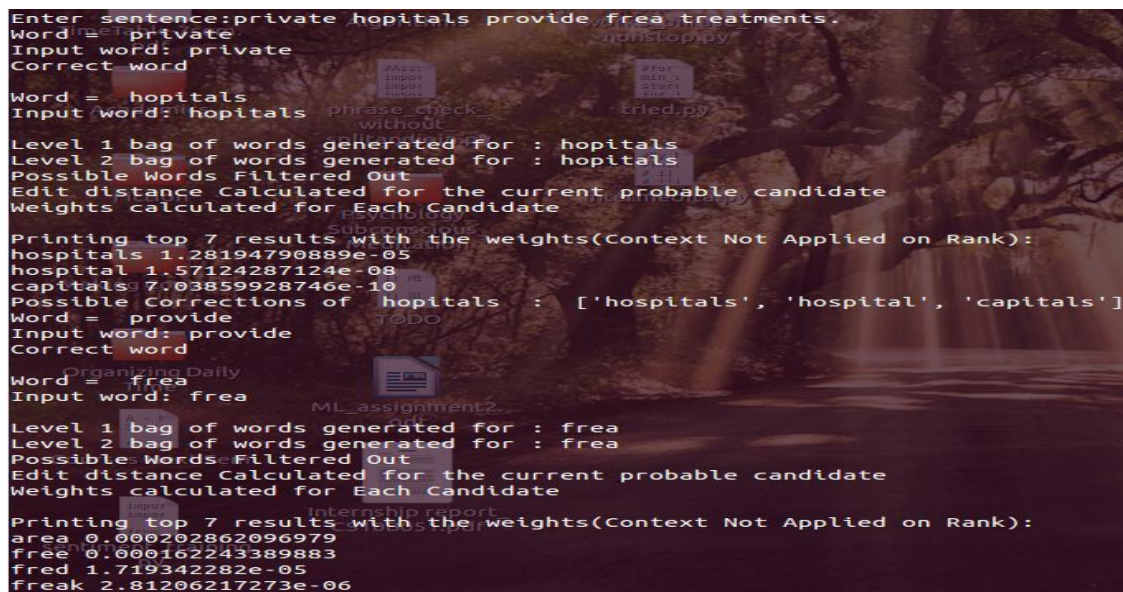
a) A similar idea is used as was used in the single word check. But the only different thing here being there are 2 set of possible word candidates for 2 words in error.

b) Now to cover all cases and get as accurate a result as possible, we pair wise check for each possibility and send the sentence with a pair of probable candidates.

Hence, for example – we have a set of 7 probable candidates each, then we send in to check for 14 sentences (7 correct for set 1 and the probable candidate set 2 + 7 correct for set 2 and the probable candidate set 1) . Thus 14 times the single word error spell check is called. The computation is still fast enough comparatively though it may lag at times due to a long sentence.

After we get all the 14 results. We pair wise find the final values (say (a,b) (when a is fixed against checking)= r and (b,a) (when b is fixed against checking) = t : final output rank = $r*t$.

c) Hence upon calling a)-q) from single error spell check, later we find all the $r*t$ and rank them according to their descending values.



```

Enter sentence: private hopitals provide frea treatments.
Word = private
Input word: private
Correct word

Word = hopitals
Input word: hopitals
Level 1 bag of words generated for : hopitals
Level 2 bag of words generated for : hopitals
Possible Words Filtered Out
Edit distance Calculated for the current probable candidate
Weights calculated for Each Candidate
Printing top 7 results with the weights(Context Not Applied on Rank):
hopitals 1.28194790889e-05
hospital 1.57124287124e-08
capitals 7.03859928746e-10
Possible Corrections of hopitals : ['hopitals', 'hospital', 'capitals']
Word = provide
Input word: provide
Correct word

Word = frea
Input word: frea
Level 1 bag of words generated for : frea
Level 2 bag of words generated for : frea
Possible Words Filtered Out
Edit distance Calculated for the current probable candidate
Weights calculated for Each Candidate
Printing top 7 results with the weights(Context Not Applied on Rank):
area 0.000202862096979
free 0.000162243389883
fred 1.719342282e-05
freak 2.81206217273e-06
  
```



```

flea 1.56241309774e-06
urea 9.37729261843e-07
Possible corrections of 'frea': ['area', 'free', 'fred', 'freak', 'freya', 'flea', 'urea']
word = 'treatments'
Input word: treatments
Correct word

Possible corrections of error words are: {'frea': ['area', 'free', 'fred', 'freak', 'freya', 'flea', 'urea'], 'hospitals': ['hospitals', 'hospita
l', 'capitals']}
ALL probable sentences to be checked: [['private', 'hospitals', 'provide', 'frea', 'treatments'], ['private', 'hospital', 'provide', 'frea', 'tr
eatments'], ['private', 'capitals', 'provide', 'frea', 'treatments']]
Measures Being Evaluated
Non stop words in the sentence include: ['private', 'hospitals', 'provide', 'treatments']
Before Magnification

free 3.93581977477e-57
freya 2.37127395647e-68
urea 2.26306668313e-68
flea 1.31504336374e-68
freak 1.17888235078e-68
fred 4.36535417749e-69
area 3.63471432536e-70
Relative Magnification Begins (if applicable)
After Magnification!

free 1.26319473571e-67
freya 2.50556283092e-79
urea 2.39122761403e-79
flea 1.3895162827e-79
freak 1.24564426312e-79
fred 4.61257086769e-80
area 3.50473720444e-81
Rank After Adding All Valid Measures
Sentence Training
py
free 6.29067956647e-80

```

```

freak 3.29501697065e-104
flea 9.94076594869e-105
urea 1.34646651323e-106
freya 7.7157111486e-107
fred 3.46182727403e-136
Measures Being Evaluated
Non stop words in the sentence include: ['private', 'hospital', 'provide', 'treatments']
Before Magnification

free 3.93581977477e-57
freya 2.37127395647e-68
urea 2.26306668313e-68
flea 1.31504336374e-68
freak 1.17888235078e-68
fred 4.36535417749e-69
area 3.63471432536e-70
Relative Magnification Begins (if applicable)
After Magnification!

free 1.26319473571e-67
freya 2.50556283092e-79
urea 2.39122761403e-79
flea 1.3895162827e-79
freak 1.24564426312e-79
fred 4.61257086769e-80
area 3.50473720444e-81
Rank After Adding All Valid Measures

free 6.29067956647e-80
area 5.41667141272e-101
freak 3.29501697065e-104
flea 9.94076594869e-105
urea 1.34646651323e-106
freya 7.7157111486e-107
fred 3.46182727403e-136

```



```

Non stop words in the sentence include: ['private', 'capitals', 'provide', 'treatments']
Before Magnification
TimeTable-7sem
Algorithms
works_phrase
nonstop.py
free 3.93581977477e-57
freya 2.37127395647e-68
urea 2.26306668313e-68
flea 1.31504336374e-68
freak 1.17888235078e-68
fred 4.36535417749e-69
area 3.63471432536e-70
Relative Magnification Begins (if applicable)
After Magnification!
Psychology
Subconscious
Meditation
free 1.26319473571e-67
freya 2.50556283092e-79
urea 2.39122761403e-79
flea 1.3895162827e-79
freak 1.24564426312e-79
fred 4.61257086769e-80
area 3.50473720444e-81
Rank After Adding All Valid Measures
Organizing Daily
Time
free 4.58086105277e-80
area 7.13995423573e-101
freak 2.04698013072e-104
flea 6.33253777349e-105
urea 1.26415096861e-105
freya 1.27734660124e-106
fred 3.46182727403e-136
ALL probable sentences to be checked: [['private', 'capitals', 'provide', 'area', 'treatments'], ['private', 'capitals', 'provide', 'free', 'treatments'], ['private', 'capitals', 'provide', 'fred', 'treatments'], ['private', 'capitals', 'provide', 'freak', 'treatments'], ['private', 'capitals', 'provide', 'flea', 'treatments'], ['private', 'capitals', 'provide', 'freya', 'treatments'], ['private', 'capitals', 'provide', 'urea', 'treatments']]
Measures Being Evaluated
Non stop words in the sentence include: ['private', 'provide', 'area', 'treatments']

```

```

hospitals 1.50104996732e-56
hospital 3.15777288609e-57
capitals 1.63440356671e-68
Relative Magnification Begins (if applicable)
After Magnification!
Academics
phrase_check
without
copy
hospitals 5.69665911313e-67
hospital 1.03250513877e-67
capitals 1.82380584535e-79
Rank After Adding All Valid Measures
Psychology
Subconscious
Meditation
hospitals 1.41760443181e-89
hospital 3.14919819579e-93
capitals 4.17590138027e-106
Measures Being Evaluated
Non stop words in the sentence include: ['private', 'provide', 'free', 'treatments']
Before Magnification
hospitals 1.50104996732e-56
hospital 3.15777288609e-57
capitals 1.63440356671e-68
Relative Magnification Begins (if applicable)
After Magnification!
Courses Next Sem
hospitals 5.69665911313e-67
hospital 1.03250513877e-67
capitals 1.82380584535e-79
Rank After Adding All Valid Measures
sentiment_training
py
hospitals 1.21200407786e-84

```

capitals 1.43626780567e-101
Measures Being Evaluated
Non stop words in the sentence include: ['private', 'provide', 'fred', 'treatments']
Before Magnification
hospitals 1.50104996732e-56
hospital 3.15777288609e-57
capitals 1.63440356671e-68
Relative Magnification Begins (if applicable)
After Magnification!
hospitals 5.69665911313e-67
hospital 1.03250513877e-67
capitals 1.82380584535e-79
Rank After Adding All Valid Measures
hospitals 7.01928465684e-102
hospital 1.55932911051e-105
capitals 1.56864651623e-118
Measures Being Evaluated
Non stop words in the sentence include: ['private', 'provide', 'freak', 'treatments']
Before Magnification
hospitals 1.50104996732e-56
hospital 3.15777288609e-57
capitals 1.63440356671e-68
Relative Magnification Begins (if applicable)
After Magnification!
hospitals 5.69665911313e-67
hospital 1.03250513877e-67
capitals 1.82380584535e-79

hospitals 1.9283977486e-90
hospital 4.28392192802e-94
capitals 2.67722469032e-107
Measures Being Evaluated
Non stop words in the sentence include: ['private', 'provide', 'freya', 'treatments']
Before Magnification
hospitals 1.50104996732e-56
hospital 3.15777288609e-57
capitals 1.63440356671e-68
Relative Magnification Begins (if applicable)
After Magnification!
hospitals 5.69665911313e-67
hospital 1.03250513877e-67
capitals 1.82380584535e-79
Rank After Adding All Valid Measures
hospitals 7.77218797971e-92
hospital 1.72658604995e-95
capitals 2.87546659354e-108
Measures Being Evaluated
Non stop words in the sentence include: ['private', 'provide', 'flea', 'treatments']
Before Magnification
hospitals 1.50104996732e-56
hospital 3.15777288609e-57
capitals 1.63440356671e-68
Relative Magnification Begins (if applicable)
After Magnification!


```

hospitals 1.50991214168e-90
hospital 3.35425911892e-94
capitals 2.14952179112e-107
Measures Being Evaluated
Non stop words in the sentence include: ['private', 'provide', 'urea', 'treatments']
Before Magnification
hospitals 1.50104996732e-56
hospital 3.15777288609e-57
capitals 1.63440356671e-68
Relative Magnification Begins (if applicable)
After Magnification!
hospitals 5.69665911313e-67
hospital 1.03250513877e-67
capitals 1.82380584535e-79
Rank After Adding All Valid Measures
hospitals 7.77218797971e-92
hospital 1.72658604995e-95
capitals 1.63071818845e-107
Final Answer Ordering is as follows:
free hospitals 7.62432928709e-164
hospital free 1.6937393462e-167
capitals free 6.57934325234e-181
area hospitals 7.6786974003e-190
area hospital 1.70581718401e-193
freak hospitals 6.35410330781e-194
flea hospitals 1.50096832036e-194

```

Sample Example: Private **hospitals** to provide **frea** treatment to the poor.

- **Single Homophone Error handling:**

Basic Idea & Implementation Details:

a) When no error is found in the word, we check for a single homophone error. (Limitation: if there are 2 homophones possible in the word, it checks for the 1st one which occurred while parsing the sentence)

b) Now it finds the set of all possible homophones from the dictionary. And sets it as the probable candidate set for the homophone word in error.

c) With this info, the single word error check steps are called and in the end, the more meaningful homophone takes its place in the final sentence based on the rank ordering.

```

Enter sentence: I went to there home. _85_3.py x sentences_phras
Word = i
Input word: i
Correct word
1065 possibleCandidatesOfIncorrect[inputWord]
1067 print "Possible Corrections of ",inputWo
1069 incorrectWordIndex[inputWord] = storeInd
1071 #store the possible word list with their
1073 index+=1
1075 #Loop ends to check each word in input
1077 Word = to
1079 Input word: to
1081 Correct word
1083 #All correct words - check for homophone ; else decl
1085 if(flag==0):
1087     homophone_flag=1
1089 Word = there
1091 Input word: there
1093 Correct word
1095 else:
1097     checkHomophones(wordsInInput)
1099 Word = home
1101 Input word: home
1103 Correct word
1105 if(flag==1 and len(wordsInInput)==1):
1107     rank = getRank(inputWord,possibleCandidates)
1109 elif(flag==1 and len(wordsInInput)>1):
1111     if(len(incorrectWordIndex)==1):
1113         Checking homophones for: ['i', 'went', 'to', 'there', 'home']
1115         Measures Being Evaluated
1117         Non stop words in the sentence include: ['went', 'home']
1119         Before Magnification
1121         elif(len(incorrectWordIndex)==2):
1123             #for 2 errors in a sentence
1125             print "Possible Corrections of error words a
1127             their 1.13079509176e-17
1129             there 4.27450713028e-44
1131             Relative Magnification Begins (if applicable)
1133             for word in possibleCandidatesOfIncorrect:
1135                 Rank After Adding All Valid Measures
1137                 if(count==1):
1139                     to_chk = word
1141                 if(count==2):
1143                     their 1.13079509176e-17
1145                     there 4.27450713028e-44
1147                     count+=1

```

Sample Example: I went to **there** home.

- Splitting of a long word into its meaningful constituents:
 - a) If no possible candidate replacements are found for an erroneous word, we try to split it across and check if a set of smaller meaningful constituents exist. If still not, then the word gets broken down into single letter/least possible sized constituents.
 - b) The working goes as follows: The word is divided into 2 words if both are in the dictionary. If I find such a scenario, then divide them into two and work with each individually to extract out smaller meaningful words of the largest length. Else if not, find the largest possible

meaningful word, split it accordingly and work only with the remaining half.

- c) Do this recursively, until all the minimal (but longest) individual components are formed.
- d) Display the result and match with the expected answer.

```
Enter sentence:footballhalloffame
Word = footballhalloffame
Input word: footballhalloffame
Level 1 bag of words generated for : footballhalloffame
Level 2 bag of words generated for : footballhalloffame
Possible Words Filtered Out
Edit distance Calculated for the current probable candidate
Weights calculated for Each Candidate
Printing top 7 results with the weights(Context Not Applied on Rank):
No probable words found
Did you forget spaces? In that case you probably meant: ['football', 'hall', 'of', 'fame']
Enter sentence:howwasthepresentation?
Word = howwasthepresentation
Input word: howwasthepresentation
Level 1 bag of words generated for : howwasthepresentation
Level 2 bag of words generated for : howwasthepresentation
Possible Words Filtered Out
Edit distance Calculated for the current probable candidate
Weights calculated for Each Candidate
Printing top 7 results with the weights(Context Not Applied on Rank):
No probable words found
Did you forget spaces? In that case you probably meant: ['how', 'was', 'the', 'presentation']
```

Sample Example: footballhalloffame; howwasyourpresentation?

- Combining a smaller set of constituents to a bigger meaningful word(s):
 - a) If no homophone errors exist and the sentence is correct, it might be the case that there exists another possibility of the sentence being in a much more structured order.
 - b) For the same, initially merger all the words in the sentence. After this, call the splitting algorithm used to split the word into longest constituents. With this you might just get much more structured format of the input sentence. Display the result. Check for both correct results to be in the expected output.

```

Enter sentence:the r e was n o snow fall
Word = the
Input word: the
Correct word
Word = r
Input word: r
Correct word
Word = e
Input word: e
Correct word
Word = was
Input word: was
Correct word
Word = n
Input word: n
Correct word
Word = o
Input word: o
Correct word
Word = snow
Input word: snow
Correct word
Word = fall
Input word: fall
Correct word
Checking homophones for: ['the', 'r', 'e', 'was', 'n', 'o', 'snow', 'fall']
Sentence was correct but sentence format may be:
['there', 'was', 'no', 'snowfall']

```

Sample Example: 'the r e was n o snow fall'

- No Error – Correct Input Sentence:

- If the sentence is correct, it will show that it is correct. Besides, as the algorithm demands, if there is another possibility of the sentence being much more structured, it will display the same as well. But the underlying fact remains that the sentence is correct. Check for both predictions with the expected result.

```

Enter sentence:The coyote fox.
Word = the
Input word: the
Correct word
Word = coyote
Input word: coyote
Correct word
Word = fox
Input word: fox
Correct word
Checking homophones for: ['the', 'coyote', 'fox']
Sentence was correct but sentence format may be:
['the', 'coyote', 'fox']

```

Sample Example: the coyote fox

Output Testing:

- a) In a while loop keep reading input test cases from the 3 test files (namely – words.tsv; phrases.tsv; sentences.tsv).
- b) The files have the erroneous words/phrases/sentence and the corrected words/phrases/sentence in one line each separated by a tab.
- c) Read the input instance, operate on it and return the expected result.
- d) Match the top 5-7 results returned with the expected answer.
- e) Apply Mean Reciprocal Rank measure to give a rank to our predicted answer.
- f) Sum up all the rank measures and divide by the total no. of test cases for each kind of input file. This gives us a performance measure for our code. The code segment to do the following matching and rank prediction is separately handled for each type of errors (homophones; splitting; single word etc) as **getRank()**.
- g) We have been provided with a test case set on moodle, and working/ testing over that set gave us the following results:

```
Total MRR of Words = 0.766666666667
Total MRR of phrases = 0.894736842105
Total MRR of sentences = 0.955555555556
```

Strengths and Weaknesses:

Strengths – 1) The measures chosen (for each type of input – word/phrase/sentence), balance well between corpora knowledge and semantic relation info and give the correct result expected in most cases. Eg: My favorite **pes** is a cat. As is seen in the snapshot: pet is initially of very low order during spell check, then corpora knowledge pushes it up and in the end semantic relatedness factor takes control and brings it out as the final correct result. The program structure for this goes as follows:

```

Enter sentence: my favorite pes is a cat.
Word = my
Input word: my
Correct word

Word = favorite
Input word: favorite
Correct word

Word = pes
Input word: pes

Level 1 bag of words generated for : pes
Level 2 bag of words generated for : pes
Possible Words Filtered Out
Edit distance Calculated for the current probable candidate
Weights calculated for Each Candidate

Printing top 7 results with the weights(Context Not Applied on Rank):
per 0.000232267511914
yes 9.03255493881e-05
peas 1.53193087783e-05
pen 1.15630138031e-05
pbs 5.93835791825e-06
des 5.93598162456e-06
pet 5.31379915044e-06
Possible Corrections of pes : ['per', 'yes', 'peas', 'pen', 'pbs', 'des', 'pet']
Word = is
Input word: is
Correct word

Word = a
Input word: a
Correct word

Word = cat
Input word: cat
Correct word

```

```

Measures Being Evaluated
Non stop words in the sentence include: ['favorite', 'cat']
Before Magnification

pbs 4.33780281903e-75
pet 2.17926375538e-75
pen 1.65804468577e-75
yes 3.51864795023e-76
peas 7.46103563093e-88
des 6.101006365758e-88
per 4.7202444173e-89
Relative Magnification Begins (if applicable)
Network
After Magnification!

pbs 2.65188249142e-86
pet 1.3322761864e-86
pen 1.01363290487e-86
yes 2.15109844361e-87
peas 1.76888047568e-99
des 1.42488051088e-99
per 1.00397633846e-100
Rank After Adding All Valid Measures

pet 3.65988248301e-92
pen 1.54385964311e-93
pbs 2.00991005951e-95
yes 2.04145790907e-97
peas 2.77473914826e-106
des 6.96807880773e-109
per 4.87205234054e-130

```

- 2) NGram model is strong for trivial case errors in the input sentence.
- 3) The powers chosen for approximation of parameter weight estimation prove to be giving good results as seen upon rigorous testing.

Weaknesses:

- 1) Dictionary Read takes a lot of time to read and extract the required info (say word_count or Ngram conversion etc.). More the corpora available, more is the time to read. This takes up a large amount to running time. But a trade-off needs to be met to get enough knowledge and do things faster at the same time.
 - 2) The Splitting and Combining of input words leads to spurious results at times based on the algorithm used.
 - 3) Smoothing and Semantic Measures can be improved upon, some results obtained from the semantic measure are very unexpected. Eg: 'spend' and 'money' have no similarity answer (WUP_similarity) but 'spending' and 'money' give some high positive measure value. Simple Laplace smoothing has been used. With more knowledge and approximation, we could have tried upon other methods of smoothing.
-