Probabilistic Non-word Word Correction using Kernighan Noisy Channel Model

An approach to Spelling Correction

Subhasis Dutta
CS 6320 Natural Language Processing (Spring 2017)
The University of Texas at Dallas
Richardson, TX
sxd150830@utdallas.edu

***Abstract* — This document describes an approach to probabilistic parsing test sentences. Identifying the words that have been misspelled and fix the spellings using Noisy Channel Model, Error Confusion Matrix and Dametau-Levenshtein Edit Distance. It also discusses some of the experiments and observations performed. Also some shortcomings of this approach and potential improvement are discussed.**

***Index Terms*—Bi-Gram Model, Kernighan Noise Channel Model, probabilistic parsing, Error Confusion Matrix, Edit Distance** *(key words)*

I. Introduction

The noisy channel model can be used to apply to the task of detecting and correcting spelling errors. In the noisy channel model, we imagine that the surface form, we see is actually a “distorted” form of an original word passed through a noisy channel. The decoder passes each hypothesis through a model of this channel and picks the word that best matches the surface noisy word. The intuition of the noisy channel model is to treat the misspelled word as if a correctly spelled word had been “distorted” by being passed through a noisy communication channel. [1]

This channel introduces “noise” in the form of substitutions or other changes to the letters, making it hard to recognize the “true” word. Our goal, then, is to build a model of the channel. Given this model, we then find the true word by passing every word of the language through our model of the noisy channel and seeing which one comes the closest to the misspelled word.

This application can take in a bunch of text which we consider to have been sent through a noisy channel. Then we process the text per sentence. For each sentence we identify the words that may have been miss-spelled by finding if it is a non-word by checking it with a dictionary. We then identify the candidates for the incorrect word by using edit-distance. Using the candidate words and confusion matrix the probability are obtained and finally using bigram probability model the word with the highest probability is chosen as the replacement for the incorrect word.

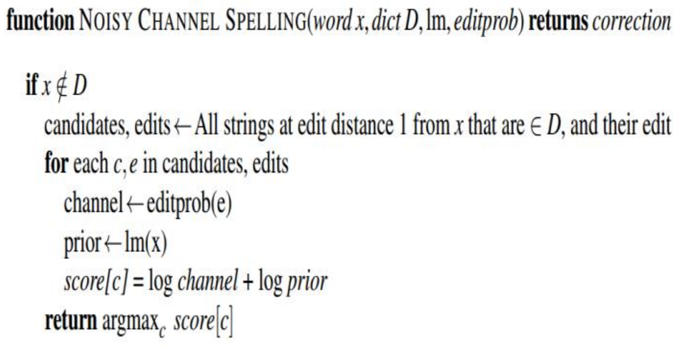
II. Related Work

This noisy channel model was applied to the spelling correction task at about the same time by researchers at AT&T Bell Laboratories (Kernighan et al. 1990, Church and Gale 1991) and IBM Watson Researchers (Mays et al. 1991).

III. Approach

For this paper, the noisy channel for correcting words in a text was implemented. A package to correct non-word spelling error in sentences using bi-gram MAP Language Models, Noisy Channel Model, Error Confusion Matrix and Damerau-Levenshtein Edit Distance was developed in Python.

The pseudo-code for the Noisy Channel Model is given as:-



```
function NOISY CHANNEL SPELLING(word x, dict D, lm, editprob) returns correction

if $x \notin D$
  candidates, edits $\leftarrow$ All strings at edit distance 1 from x that are $\in D$, and their edit
  for each c, e in candidates, edits
    channel $\leftarrow$ editprob(e)
    prior $\leftarrow$ lm(x)
    score[c] = log channel + log prior
  return $\text{argmax}_c$ score[c]
```

Fig. 1: Algorithm for noisy channel model for spelling correction for unknown words [1]

The process consist of the following steps:

1. Ingesting a text document and separating each sentence. For this a simple split by “.” was used. Each of the sentence is processed separately. They can be done parallel to make the execution faster as the execution for each sentence is independent but for simplicity we have processed it sequentially.
2. For each sentence we extract the tokens, again for simplicity we only consider checking only words that

consist of characters in a-z range. For this we ignore all numerical character words and convert all words to lower case.

3. Iterate over each acceptable token, and then check if it is a valid or in-valid word by doing a fast lookup on an English dictionary.
4. If the word is found to be a non-word we look for its candidates. All words that are at an edit distance of one are considered as candidate words. To find the candidates we use Dametau-Levenshtein Edit Distance.
5. For each candidate word we identify what kind of edit will match the non-word. Using a simple string comparison we identify the edit type as Insertion, Deletion, Substitution or Transposition and the characters involved in the candidate and non-word.
6. Based on the edit type identified, we look up the appropriate confusion matrix and obtain the count from a sample data corpus and find the channel probability for the occurrence of such an edit.
7. After that by considering the candidate word and the words before and after the word we use a bigram model and compute the sentence probability.
8. Using the sentence probability for all of the sentence made by each of the candidate word we chose the word as a replacement that gives the largest probability in the bi-gram model.
9. Using a simple join after checking for the case of the old word we transform and create the fixed sentence and provide the output result.

IV. DATA SOURCE

The following sources were considered for obtaining the data required to implement this project: -

- English word data corpus – For the corpus to obtain the count of occurrence of character sequence we used a large collection of English stories from Shakespeare and other English novels obtained from Python NLTA and Wikipedia. [2] [3]
- English Dictionary file – This is the file obtained containing all valid English words from NLTK data corpus. [3]
- Deletion Confusion Matrix – A count matrix that consist of count from a large text when “x” is removed followed by “y”.
- Insertion Confusion Matrix - A count matrix that consist of count from a large text when “x” is added followed by “y”.
- Substitution Confusion Matrix - A count matrix that consist of count from a large text when “x” is replaced by “y”.
- Transposition Confusion Matrix - A count matrix that consist of count from a large text when “x” is switched with by “y” side by side.

V. IMPLEMENTATION DETAILS

This section describe the input expected to the project, the design choices, the expected results and the choice of various tools and languages.

A. Input Expected

As there are two phases to the implementation of this project: one the python script to generate the dictionary and probability count matrix for bigram probability in the memory, and second to parse the text and identify non-words and process the sentence. So, the first phase expects python and NLTK library to be installed into the system where the python script has to be run. This program detects the installation of NLTK in the system as the input and generates all the required matrix and dictionary in the memory. In the next phase, the generated matrix for bi-gram probability and the loaded confusion matrix are used to parse each sentence and find the correct replacements and for the given sentences its correct sentence is displayed to the user.

B. Design choices

- A third party library, Python Edit Distance version 0.3.1 [4] was used to find the edit distance using Levenshtein distance.
- A separate class was defined to parse the corpus and find Bi-Gram Probability (BiGramService).
- A class was defined to create a concept of a probability matrix, and parse each candidate and non-word and use the confusion matric (WordCorrectionService)
- A class was defined to take in a text break them into sentence and using Word Correction Service identify and correct words and return the fixed text.
- All the code was designed to run through a driver program which provided a base to input and display the output. If a sentence is provided to the parser has either invalid or unrecognized by the grammar, the parser tries to fix it and returns a user-friendly message specifying the corrected text same.

C. Expected Results

The following results are obtained out of the project: -

- The text is converted to have the same case sensitivity.
- All non-word that are identified are replaced with the most probable candidate word and returned.
- The output also says which words it identified as correct and which it decided to change and what action was taken to convert the non-word to a dictionary word.

D. Other Implementation details

- Languages: Python 2.7 (For parsing of data and building model and executing the text)
- Environment configuration: 6GB RAM, Intel Core i5 processor with operating system as Ubuntu 15.10
- Size of English dictionary file – 1128 KB
- Size of word corpus file -26122 KB
- Size of confusion matrix – 7 KB each
- IDE used – Pycharm from Jet Brains (For Python)
- Time taken for implementation – 20 hours

VI. Experiments and observations

This section discusses about the various parameters of the project and experiments conducted on them and observations made along the process.

A. Parameters

The following parameters were observed on the experiments conducted.

1) Running time of the parser – In Seconds.
2) Size of the grammar – In KB.
3) Length of the sentence provided by the user – In number of words (All strings separated by space).
4) Running time – Compared to more valid and invalid cases

B. Experiments

Firstly, the sentences was used to validate the working of the noisy channel model. Once the parser was thoroughly tested, more complex sentences were used to conduct experiments by varying each of the parameters discussed above and recording the observations made for each experiment.

Experiments were made in the following categories: -

- Running time was noted as size of the sentences was varied
- Running time was noted as the probability of having candidate words for a non-word was varied
- Accuracy and recognition percent was noted with the size of the length of the sentences and edit distance from candidate words.
- Sentence probability was noted as length of sentence was varied.

C. Observations

With the experiments performed as discussed, the following observations were made: -

TABLE I. Performance compared to size Number of Tokens

Test case	**Runtime (In Seconds)**			
	Token	***Non-word Tokens***	***No. of runs***	***Run Time***
Test 1	16	5	5	5.05
Test 2	14	3	5	0.83

For the above experiments in table 1, almost same sentence with different wrong tokens was chosen and the run times were compared. It is clear that majority of time was spent on finding the correct replacement for the non-word from a list of candidate word rather than on identifying if the spelling of a word is correct or wrong.

TABLE II. Performance compared to Choice of candidate words

Non-Word	**Runtime (In Milliseconds)**		
	Candidates	***No. of runs***	***Run Time***
briliant (1)	brilliant	5	207
acress (6)	'actress', 'acres', 'access', 'cress', 'across', 'agress'	5	1283
wno (17)	'wao', 'ono', 'no', 'vno', 'wro', 'wgo', 'who', 'wnn', 'wo', 'woo', 'wpo', 'bno', 'wnt', 'wyo', 'wnx', 'wco', 'uno'	5	2858
tweve (2)	'twelve', 'twere'	5	634

From the above experiments in table 2, it can be observed that the running time is comparable to the no of candidate words found for a word. If more candidates are found it takes more time to execute.

TABLE III. Accuracy and recognition

Grammar	**Accuracy (In %)**			
	Wrong Words	***Found Correct Replacement***	***Wrong Replacement***	***Accuracy***
Test 1	1	4	1	14
Test 2	915	1	1	96

For the above experiments in table 3, accuracy may not be the correct measurement for judging the accuracy but analysis of the results do give the strengths and weakness of the model.

An analysis of each of the non-word is given below:

1. **Brilliant** – Correctly identified as brilliant with
   Edit Type: **brilliant**: ('DELETION', 'l', '', 'i', 'il')
2. **Acress** – Candidates identified ['actress', 'acres', 'access', 'cress', 'across', 'agress']
   Correctly identified as actress with
   Edit Type: **actress**: ('DELETION', 't', '', 'c', 'ct')
3. Wno – Candidates identified ['wao', 'ono', 'no', 'vno', 'wro', 'wgo', 'who', 'wnn', 'wo', 'woo', 'wpo', 'bno', 'wnt', 'wyo', 'wnx', 'wco', 'uno']
   Incorrectly identified as wno. Correct "won" not identified as it was not present in corpus.
4. **Beaiful** – Candidate identified None
   Correct "beautiful" not identified as it has edit distance of 2 which is not considered as candidate key for this experiment.

D. Result

The following input was converted to output:

1. Input - She is a **briliant acress**. She **wno Tweve** Oscar awards but her latest **mvie** is bad.
   Output - She is a **brilliant actress**. She **wao Twelve** Oscar awards but her latest **movie** is bad.
2. Input - She is a **beaiful** actress. Her **moveie** did very well at the box **ofice**.
   Output - She is a **beaiful** actress. Her **movie** did very well at the box **office**.

E. Conclusions

After recording all the observations and analyzing the results, the following conclusions were made: -

- There is a tradeoff between running time and relevance of the results generated by the parser. A short sentence with less number of candidate words was quickly processed.

- Most of the tokens that were identified as correct the ones that wer not identified was due to lack of the correct word in the corpus or the restriction on the model to consider only candidate words with edit distance 1.
- Hence, as per different scenarios, it is wise say that even though the noisy channel is very effective in finding out the correct word. And is also able to identify the correct word based on most probable candidate based on bi-gram probability on the sentence. It still needs improvements to handle more scenarios.

VII. SHORTCOMINGS AND IMPROVEMENTS

Following are some of the shortcomings and potential improvements proposed [5] [6]: -

- The current implementation cannot identify candidate the words which have more than one edit distance. While this can be easily be included in this implementation. It increases the runtime by a large extent and will make it ineffective in real world spelling correction systems.
- The package cannot detect dependency if the dependency word are not adjacent. As we are using bi-gram model. For considering more words we will have to use a large n-gram model which will make the system more complex and increase the computation time.
- The grammar corpus is also small of only 26 MB from mostly novel stories which did not cover all domains of words. For a more effective spell corrector more domains with more different words could be used.

ACKNOWLEDGMENT

I wish to acknowledge Professor for the Natural Language Processing class, Dr. Dan Moldovan for approving the project proposal and providing the guidelines for the project. Also, I acknowledge the teaching assistant for the class, Linrui Zhang, for resolution of queries and validating the demo run of the project.

REFERENCES

[1] Spelling correction and the Noisy Channel by Jurafsky, Daniel https://web.stanford.edu/~jurafsky/slp3/5.pdf

[2] Wikipedia Noisy channel model https://en.wikipedia.org/wiki/Noisy_channel_model

[3] Python NLTK http://www.nltk.org/

[4] Python Edit Distance https://pypi.python.org/pypi/editdistance

[5] Part of Speech Tagging by Moldovan, Dan http://www.hlt.utdallas.edu/~moldovan/CS6320.17S/Lecture%205%20Part-of-Speech%20Tagging.pdf

[6] How to write a Spelling Corrector, Norvig Peter http://www.norvig.com/spell-correct.html