Bansilal Ramnath Agarwal Charitable Trust's

Vishwakarma Institute of Technology, Pune-37

(An autonomous institute of Savitribai Phule Pune University)

Statistical Inference Lab Assignment 4

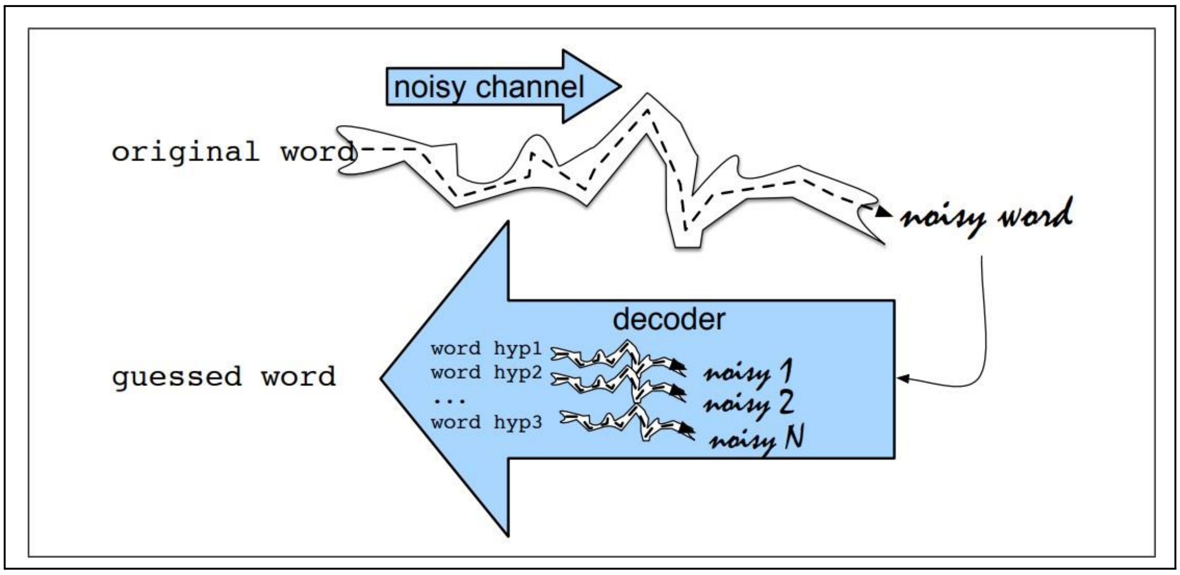
Year	Third
Branch	AI&DS
Division	AI-A
Batch	2
PRN	12210881
Roll no.	44
Name	Vedant Deore

Problem Statement:

Noisy channel model

Noise Channel Model Overview

The Noise Channel Model is a probabilistic model that assumes any observed misspelled word is a "noisy" version of a correctly spelled word. The idea is rooted in communication theory, where information (in this case, the correct word) is transmitted through a channel that adds noise (errors like substitutions, deletions, insertions, or transpositions), resulting in a potentially distorted version of the original word. The model is commonly used in spelling correction, where the task is to determine the most likely original word given the noisy, observed version.



Theoretical Foundation

The model is based on Bayesian inference, where the goal is to find the word w in a vocabulary V that maximizes the probability of having produced the observed misspelled word x. This is mathematically represented as:

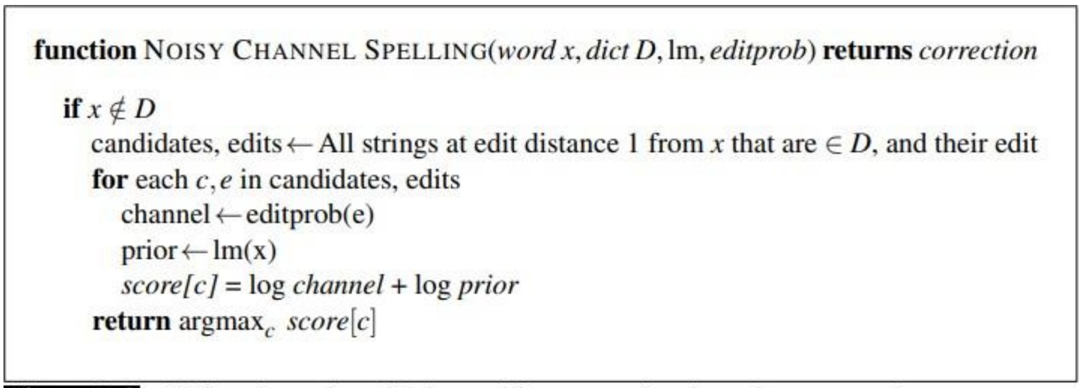
ŵ = argmax_{w ∈V} P(w | x)

Using Bayes' Theorem, this can be expanded to:

ŵ = argmax_{w ∈V} P(x | w) ·P(w)

P(x | w) is the likelihood or the channel model that represents the probability of the word w being distorted into x.

P(w) is the prior probability of the word w occurring in general (often derived from a language model).



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

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

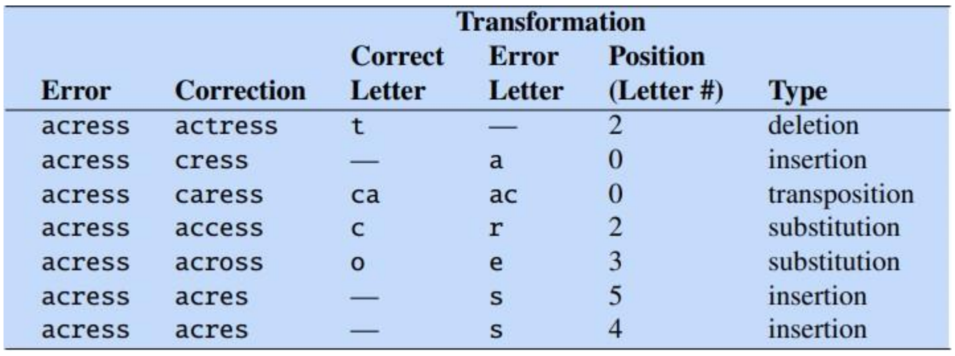
Figure B.2 Noisy channel model for spelling correction for unknown words.

Practical Application in Spelling Correction

In a spelling correction task:

- Generate Candidates: For any misspelled word x, generate a list of possible correct words (candidates) that are similar in spelling.
- Score Candidates: Each candidate word is scored using the noisy channel model. This involves calculating P(x | w), which estimates the probability that the correct word w was distorted into the observed word x, and multiplying it by P(w), the prior probability of the word.
- Select the Best Candidate: The candidate word with the highest score is selected as the most likely correct word.

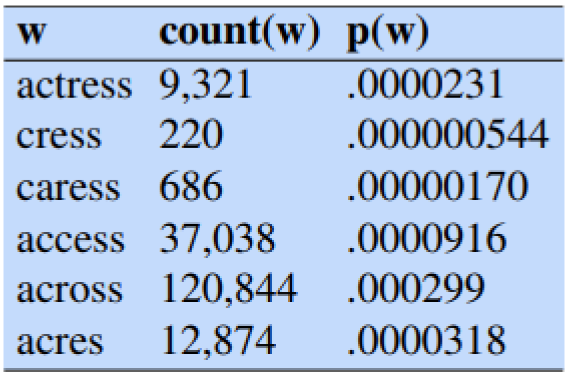
Example Calculation



		Transformation			
Error	Correction	Correct Letter	Error Letter	Position (Letter #)	Type
acress	actress	t	—	2	deletion
acress	cress	—	a	0	insertion
acress	caress	ca	ac	0	transposition
acress	access	c	r	2	substitution
acress	across	o	e	3	substitution
acress	acres	—	s	5	insertion
acress	acres	—	s	4	insertion

Consider the misspelled word "acress" and possible corrections "actress", "access", "across", etc. The model would compute the likelihood P(x | w) for each candidate by considering the type of error (e.g., deletion of a letter, substitution of one letter for another) and use a precomputed confusion matrix to estimate these probabilities. The prior P(w) would typically be based on the frequency of these words in a large corpus.

For example:



w	count(w)	p(w)
actress	9,321	.0000231
cress	220	.000000544
caress	686	.00000170
access	37,038	.0000916
across	120,844	.000299
acres	12,874	.0000318

Likelihood P(x | w): For "actress", if the error was a deletion of 't', the likelihood might be calculated based on how often 't' is mistakenly omitted.

Prior P(w): The word "actress" might have a higher prior if it is more common in the language.

Finally, the model multiplies these two probabilities and selects the word with the highest resulting value as the correction.

Practical Implementation

When implementing the Noise Channel Model in a practical session:

1. Data Preparation: Collect a list of common spelling errors and their correct forms. You can also create a confusion matrix from these examples.
2. Candidate Generation: Use algorithms like Damerau-Levenshtein distance to generate candidate corrections with a minimal number of edits.
3. Scoring: Calculate the likelihood and prior probabilities for each candidate using the noisy channel model formula.
4. Selection: Choose the word with the highest combined probability as the correct spelling.

Algorithm for Spell Checker Implementation

1. Build Corpus

1.1. Input: Corpus text.
1.2. Process:
- Convert the entire corpus text to lowercase.
- Use regular expressions to find all word tokens.
- Count the frequency of each word using a Counter from the collections module.

1.3. Output: A Counter object representing the word frequency in the corpus.

2. Calculate Word Probability

2.1. Input: A word, the word frequency counter (corpus_counter), and the total number of words in the corpus.
2.2. Process: Calculate the probability of the word occurring in the corpus by dividing the word's frequency by the total word count.
2.3. Output: The probability value of the word.

3. Generate Edits

3.1. Input: A word (misspelled word).
3.2. Process:
- Create possible splits of the word.
- Generate all possible edits that are one edit away from the original word. These edits include:
  a) Deletes: Removing one character from the word.
  b) Transposes: Swapping two adjacent characters in the word.
  c) Replaces: Replacing each character in the word with every possible letter.
  d) Inserts: Inserting every possible letter at each position in the word.

3.3. Output: A set of all possible words that are one edit away from the original word.

4. Identify Known Words

4.1. Input: A set of words generated from the edits and the word frequency counter (corpus_counter).
4.2. Process: Check which of the generated words exist in the corpus.
4.3. Output: A set of known words that exist in the corpus.

5. Generate Candidates

5.1. Input: A misspelled word and the word frequency counter (corpus_counter).
5.2. Process:
- Generate a list of possible candidate corrections by:
  a) Checking if the word is known (exists in the corpus).
  b) Checking for known words that are one edit away.
  c) Checking for known words that are two edits away.
- Return the first non-empty set of candidates.

5.3. Output: A set of candidate words.

6. Correct the Word

6.1. Input: A misspelled word, the word frequency counter (corpus_counter), and the total number of words in the corpus.
6.2. Process:
- Generate a list of candidate corrections.
- For each candidate, calculate its probability based on the corpus.
- Select the candidate with the highest probability as the correct word.

6.3. Output: The most probable correct word.

7. Spell Checker Execution

7.1. Input: The corpus text and a misspelled word.
7.2. Process:
- Build the corpus using the provided text.
- Calculate the total number of words in the corpus.
- Generate all possible edits (one and two edits away) and identify the known words.
- Find the most probable correct word from the candidates.

7.3. Output: The corrected word and details of possible edits.

Code:

Importing Regular Expression and Counter

```
import re

from collections import Counter
```

```
def build_corpus(corpus_text):

    """

    Build a word frequency counter from the corpus text.

    """

    words = re.findall(r'\w+', corpus_text.lower())

    return Counter(words)
```

```
def probability(word, corpus_counter, total_words):

    """

    Calculate the probability of a word given the corpus counter

    and total word count.

    """

    return corpus_counter[word] / total_words if total_words > 0 else 0
```

```
def edits1(word):
    """
    Generate all possible edits that are one edit away from the
    given word.
    """
    letters = 'abcdefghijklmnopqrstuvwxyz'
    splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]
    deletes = [L + R[1:] for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R) > 1]
    replaces = [L + c + R[1:] for L, R in splits if R for c in letters]
    inserts = [L + c + R for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)
```

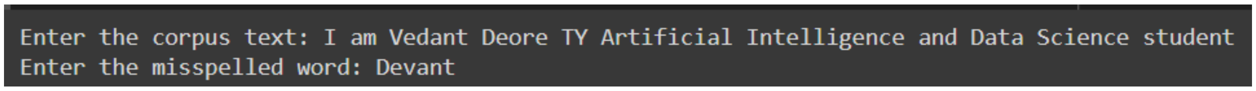
```
def known(words, corpus_counter):
    """
    Return the subset of words that are actually in the corpus.
    """
    return set(w for w in words if w in corpus_counter)
```

```
def candidates(word, corpus_counter):
    """
    Generate possible correction candidates for a given word.
    """
    known_words = known([word], corpus_counter)
    known_edits1 = known(edits1(word), corpus_counter)
    known_edits2 = known([e2 for e1 in edits1(word) for e2 in edits1(e1)], corpus_counter)
    return known_words or known_edits1 or known_edits2 or [word]
```

```
def correct(word, corpus_counter, total_words):
    """
    Find the most probable correct word from the candidates.
    """
    candidates_list = candidates(word, corpus_counter)
    return max(candidates_list, key=lambda w: probability(w, corpus_counter, total_words))
```

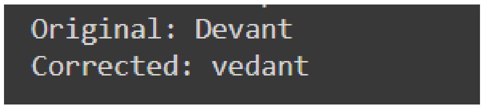
```
def spell_checker(corpus_text, misspelled_word):
    """
    Correct the given misspelled word using the provided corpus
    text.
    """
    corpus_counter = build_corpus(corpus_text)
    total_words = sum(corpus_counter.values())
    all_edits1 = edits1(misspelled_word)
    known_edits1 = known(all_edits1, corpus_counter)
    all_edits2 = {e2 for e1 in all_edits1 for e2 in edits1(e1)}
    known_edits2 = known(all_edits2, corpus_counter)
    final_correction = correct(misspelled_word, corpus_counter, total_words)
    return final_correction, {
        'all_edits1': all_edits1,
        'known_edits1': known_edits1,
        'all_edits2': all_edits2,
        'known_edits2': known_edits2
    }
```

```
# Input from the user

corpus_text = input("Enter the corpus text: ")

misspelled_word = input("Enter the misspelled word: ")
```



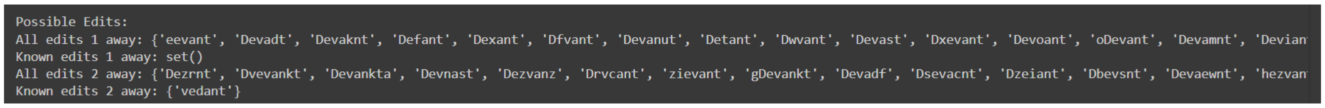
```
Enter the corpus text: I am Vedant Deore TY Artificial Intelligence and Data Science student
Enter the misspelled word: Devant
```

```
# Correcting the misspelled word

corrected_word, edits_info = spell_checker(corpus_text, misspelled_word)



print(f"Original: {misspelled_word}")

print(f"Corrected: {corrected_word}")
```



```
Original: Devant
Corrected: vedant
```

```
# Print all possible edits

print("\nPossible Edits:")

print(f"All edits 1 away: {edits_info['all_edits1']}")

print(f"Known edits 1 away: {edits_info['known_edits1']}")

print(f"All edits 2 away: {edits_info['all_edits2']}")

print(f"Known edits 2 away: {edits_info['known_edits2']}")
```



```
Possible Edits:
All edits 1 away: {'eevant', 'Devadt', 'Devaknt', 'Defant', 'Dexant', 'Dfvant', 'Devanut', 'Detant', 'Dwvant', 'Devast', 'Dxevant', 'Devoant', 'oDevant', 'Devamnt', 'Devian
Known edits 1 away: set()
All edits 2 away: {'Dezrnt', 'Dvevankt', 'Devankta', 'Devnast', 'Dezvanz', 'Drvcant', 'zievant', 'gDevankt', 'Devadf', 'Dsevacnt', 'Dzeiant', 'Dbevsnt', 'Devaewnt', 'hezvan
Known edits 2 away: {'vedant'}
```