First-two-char input method

Orcun Ilker Doeger

1. Introduction

Efficient text input is essential nowadays, particularly on devices with small keyboards like smartphones. Traditional typing can also be slow and prone to mistakes, which makes finding faster and more accurate input methods even more important. One approach is the first-two-char input method, where only the first two characters of each word are needed, and the system predicts and completes the word. This project focuses on implementing this method using KyTea and Python.

We will train a model using a dataset of word pairs, each consisting of the first two characters and the full word. This report outlines the steps taken to prepare the dataset, train the model, and test its performance. By evaluating the model's accuracy, we aim to demonstrate the effectiveness of the first-two-char input method in improving typing efficiency in a statistical manner.

2. Method

1. **Training data:** Some training data is needed to be able to train a model. For that, we prepare the data with Python, that we then can feed into KyTea. The result is on the right:

```python
# Makes the training data out of the whole book (book.txt)
# Project -> Pr/Project and so on
# saves it into "book_processed_train.txt"

def prepare_text_for_training(text):
    # Split text int owords
    words = text.split()

    # Clean each word and bring it into form Pr/Project
    processed_words = []
    for word in words:
        clean_word = re.sub('[^a-zA-Z]', '', word) #only alphabetical
        if len(clean_word) > 1:
            processed_word = f"{clean_word[:2]}/{clean_word}"
        elif len(clean_word) == 1:
            processed_word = f"{clean_word}/{clean_word}"
        if processed_word:  # only if existend
            processed_words.append(processed_word)

    processed_words += "\n"
    return processed_words

# Do it
# Read
with open('book.txt', 'r') as file:
    text = file.read()

#Process
processed_words = prepare_text_for_training(text)

# Write
with open('book_processed_train.txt', 'w') as file:
    file.write('\n'.join(processed_words))

```

```
to/to
nu/number
my/my
da/days
th/that
I/I
ma/may
ap/apply
my/my
he/heart
un/unto
wi/wisdom
Sa/Saturday
Ma/March
Ma/March
My/My
un/uncle
re/returned
ho/home
an/and
wi/with
hi/him
my/my
au/aunt
Bl/Bly
```

2. **Testing data:** To test the model, we prepare a 2-gram-word text out of normal text.

```python
# Prepares the test text that i took out of the book
# Removes any non alphabetical characters and makes 2-grams of the words
# for testing

def generate2gramText(input_file, output_file):
    # Read test input
    with open(input_file, 'r') as file:
        text = file.read()

    words = text.split()

    processed_words = []
    for word in words:
        clean_word = re.sub('[^a-zA-Z]', '', word)
        if clean_word:
            processed_words.append(clean_word[:2])

    # Join the processed words with a space
    result = ' '.join(processed_words)
    result += "\n"

    # Write
    with open(output_file, 'w') as file:
        file.write(result)


input_file = 'testing_input.txt'
output_file = 'testing_input_processed_test.txt'

generate2gramText(input_file, output_file)
```

```
Sunday 8. The Lord was pleased to call her to himself. The day before
she seemed much better. But in the evening she fell worse again, and
this morning, about nine, departed. When word came of this, I was not
shocked at first; but in a while I began to reason on what had passed
between us the evening before. She then said, “When death seemed nigh
some nights since, the enemy thrust sore at me, and said, This will be
thy end: But I said, Not without God’s permission: He can bring me down
to the belly of hell, and bring me up again, as he did _Jonah_ out of
the whale’s belly. But it is one thing to talk of death, and another
```

```
Su Th Lo wa pl to ca he to hi Th da be sh se mu be Bu in th ev sh fe wo ag an th mo
ab ni de Wh wo ca of th I wa no sh at fi bu in a wh I be to re on wh ha pa be us th
ev be Sh th sa Wh de se ni so ni si th en th so at me an sa Th wi be th en Bu I sa No
wi Go pe He ca br me do to th be of he an br me up ag as he di Jo ou of th wh be Bu
it is on th to ta of de an an to ha it br ho to us Th he hu an ch an fr ca ne an so
se to ov he Bu sh sa Go is ab to bl an ke th wi me if he is pl to ta me An if he is
```

3. Training and Testing with KyTea

Using following commands, we can now develop a model with the training data which we can then use to predict the words from the 2-gram-word-text.

train-kytea -full book_processed_train.txt -nows -model train.model

```
orcun@MacBook-Air-von-Orcun python % train-kytea -full book_processed_train.txt -nows -model train.model
Scanning dictionaries and corpora for vocabulary
Reading corpus from book_processed_train.txt  done (90324 lines)
Building dictionary index done!
Creating tagging features (tag 1) done!
Training local tag classifiers done!
Printing model to train.model done!
```

kytea -model train.model -out tags < testing_input_processed_test.txt > test.out

```
orcun@MacBook-Air-von-Orcun python % kytea -model train.model -out tags < testing_input_processed_test.txt > test.out
```

4. **Evaluation**: After having a test-result (the predictions), we can now compare the resulting text to the ground-truth text:

```python
# This compares the test text (ground-truth) to the kytea model result (from the 2grams)

def read_clean_text(input_file):
    with open(input_file, 'r') as file:
        text = file.read()

    cleaned_text = re.sub(r'[^a-zA-Z\s]', '', text).lower() # Clean text

    words = cleaned_text.split() # Split to words
    return words

def compare_texts(input_file1, input_file2):
    # Clean both result and ground-truth texts
    words1 = read_clean_text(input_file1)
    words2 = read_clean_text(input_file2)

    matched_words = 0
    if len(words1) == len(words2):
        for i in range (0, len(words1)):
            if words1[i] == words2[i]:
                matched_words += 1

    percentage = 100 * matched_words/len(words1)
    return percentage


input_file1 = 'testing_input.txt'
input_file2 = 'test.out'

percentage = compare_texts(input_file1, input_file2)
print(f"Correctness: {percentage:.2f}%")
```

	words1	words2
	0	0
0	sunday	sunday
1	the	the
2	lord	lord
3	was	was
4	pleased	pleased
5	to	to
6	call	can
7	her	her
8	to	to
9	himself	his
10	the	the
11	day	day
12	before	be
13	she	she
14	seemed	see
15	much	much
16	better	be
17	but	but
18	in	in
19	the	the
20	evening	every

3. Language Resource

For the training data, a raw-text book has been used: https://www.gutenberg.org/ebooks/73797
It is consisting of 90500 words.

Author	Wesley, John, 1703-1791
Title	The works of the Rev. John Wesley, Vol. 13 (of 32)
Original Publication	Bristol: William Pine, 1771.
Credits	Richard Hulse and the Online Distributed Proofreading Team at https://www.pgdp.net (This file was produced from images generously made available by The Internet Archive)
Language	Englisch
LoC Class	BX: Philosophy, Psychology, Religion: Christianity: Churches, Church movements
Subject	Theology -- Early works to 1800
Subject	Methodist Church
Subject	Theology -- History -- 18th century
Category	Text
EBook-No.	73797
Release Date	08.06.2024
Copyright Status	Public domain in the USA.

4. Experimental Evaluation

For the testing of the model, a short part of the book consisting of 350 words has been used. This experiment yields an accuracy of 58.17%:

Correctness: 58.17%

20	evening	20	every
21	she	21	she
22	fell	22	fear
23	worse	23	would
24	again	24	again
25	and	25	and
26	this	26	the
27	morning	27	more
28	about	28	about
29	nine	29	night
30	departed	30	dear
31	when	31	when
32	word	32	would
33	came	33	can
34	of	34	of
35	this	35	the
36	i	36	i
37	was	37	was
38	not	38	not
39	shocked	39	she
40	at	40	at
41	first	41	find
42	but	42	but
43	in	43	in
44	a	44	a
45	while	45	which

5. Conclusion

This implementation of the first-two-char input method using KyTea and Python achieved an accuracy of 58.17% with this specific training and test data sets. While this shows that the model can predict full words from the first two characters to some extent, there is substantial room for improvement.

The moderate accuracy proposes that predicting words based on just the first two characters is challenging, due to ambiguities and the limited context. Several factors could have influenced the model's performance, such as the size and diversity of the training dataset or the complexity of the language.