(Short -) Exercise 4
Statistical Learning Theory (Lab)

Release: June 5, 2019 Submit: June 13, 2019.

In this exercise you will learn about a fun application of the *Naive Bayes* classifier: Spelling correction. For the informal and vivid character of natural language, the correction of syntactic and semantic errors in written language is an extremely hard task. Here you will challenge this problem by applying one of the simplest classification algorithms, namely *Naive Bayes* and see how it performs.

Notes:

- Ask for help, if needed.
- Implement does **not** mean import!
- Submit **working** code, e.g. a Jupyter Notebook.

1 Problem statement

We frame spelling correction as a classification problem and use Bayesian inference to obtain results. Although the basic problem statement is straightforward, there is plenty of room to improve the here applied models, initially published by Peter Norvig.

Given a word $w \in W$ from a corpus of words W. Find the most likely correction $c^{(k)}$ from the set of all correction candidates $C_w \subset W$ for w. We can formalize this task as follows:

$$\arg\max_k \; P(c^{(k)}|w) \tag{1}$$

By using Bayes' theorem we obtain:

$$\begin{aligned} \arg\max_k \; P(c^{(k)}|w) &= \arg\max_k \; \frac{P(w|c^{(k)})P(c^{(k)})}{P(w)} && (2) \\ &= \arg\max_k P(w|c^{(k)})P(c^{(k)}) && (3) \end{aligned}$$

We call $P(c^{(k)})$ the **language model**. The set of Candidates C_w for a given word w needs to be calculated and requires a model of typographical errors. We call $P(w|c^{(k)})$ the **error model** or **noise channel model**. You can find a more detailed explanation in [JM00] (see the pdf in the repository).

The git repository contains the following documents, that you need to complete the tasks:

- Chapter 5 of [JM00].
- *data.txt*
- *addconfusion.data*
- *delconfusion.data*
- *revconfusion.data*
- *subconfusion.data*
- *validate.csv*

2 Language Model

There are multiple classes of language models (see. [Lan]). The simplest class of model is the *unigram* model, where the probability of each word is fixed and can be obtained by using the frequency of its occurrence in a given corpus.

Problem 1. The file *data.txt* contains the *Project Gutenberg* ebook of *The Adventures of Sherlock Holmes* by *Sir Arthur Conan Doyle.* This source shall provide the basis for obtaining the corpus and for training a language model. Implement a language-model function to evaluate $P(w)$. You must load the file *data.txt*, filter the alphabetic expressions (hint: use regex) and calculate the frequencies of occurrences of the existing words.

What are the ten most frequent words and what are their respective probabilities of occurrences?

3 Correction Candidates

Real-world-spelling-errors take multiple shapes. There are typographical errors like insertions, deletions and transpositions of letters, and there are cognitive errors where the writer substitutes a wrong spelling of a homophone or near-homophone (e.g., dessert for desert, or piece for peace). The detection of the latter require context information and therefore are hard to detect. Thus we focus on the former, which are errors that were produced by the following operations:

- transposition (e.g. ”caress” for ”actress” : ”ac” → ”ca”)
- deletion (”acress” for ”actress”: missing ”t”)
- substitutions (”acress” for ”access”: substituted ”r” for ”c”)
- insertions (”actresss” for ”actress”: added ”s”)

Given a word w, we create a set of possible candidates C_w by applying all the possible edit operators on w. By concatenating operations, you can transform each word into any other arbitrary word. Yet, to keep things simply we only consider candidates $c^{(k)}$ with only one edit distance away from w. For instance, the correction candidates for the word *text*, could be:

$$C_{nie} = \{die, lie, pie, tie, nice, nine, in\}$$

Problem 2. Implement a function that creates a set of **all** possible correction candidates C_w for a given word w, that are one edit distance away from w. You do not need to concatenate operations! To make things even simpler:

- You can ignore unknown words (words, that do not occur in our corpus)
- If none of the candidates is known, return the word itself.

What is the set of correction candidates for the word *frod*?

4 Error model

Chapter 5 of [JM00] describes the *noise channel model.* Before we implement that, let us consider an even simpler error model where:

$$P(w|c^{(k)}) = P(c^{(k)}) \tag{4}$$

Problem 3. Implement a function correction(word), that returns the most probable correction candidate, in case the word is unknown. Additionally for the first version, assume:

- If w is a known word, then $P(w) > P(c^{(k)})$ and therefore return w.
- If w is unknown, calculate $\arg\max_k P(c^{(k)})$ and return $c^{(k)}$.

- If both w and its corrections are unknown, return w.

Problem 4. The file *validate.csv* contains a list of evaluation data, where each row contains a sample of the form:

right, misspelled

Use the validation data set to test your first version of the spelling correction function.

What is your success rate?

5 The Noise Channel Model

By analyzing the output of your spelling corrector, you find examples where the corrected word is indeed a real word, but not the expected one. This phenomena occurs due to the simplistic nature of our language model. *Unigram* models do not include context sensitive information. Thus, improving the language model should naturally improve the validation results.

Chapter 5. [JM00] introduces another model to improve the success rate. Instead of treating each error equally, we now consider the probability of error occurrences.

Problem 5. Implement the *noise channel model* and use the confusion matrices included in the repository. Revalidate your correction function and report your success rate. Again you can confine your functions to only consider known words with edit distance one. You can load the matrices via:

```
import ast
with open('addconfusion.data', 'r') as file:
        addmatrix=data=ast.literal_eval(file.read())
with open('subconfusion.data', 'r') as file:
        submatrix=data=ast.literal_eval(file.read())
with open('revconfusion.data', 'r') as file:
        revmatrix=data=ast.literal_eval(file.read())
with open('delconfusion.data', 'r') as file:
        delmatrix=data=ast.literal_eval(file.read())
```

Have fun!

References

[JM00] Jurafsky, Daniel ; Martin, James H.: *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall PTR, 2000

[Lan] Language model: *Language model — Wikipedia, The Free Encyclopedia.* https://en.wikipedia.org/wiki/Language_model