Lab 2 (Language Identification)

Status: Completed
Your identity is visible during marking.
Marks: 7 / 7

Submission deadline: 30 Aug 2019 23:55, 45 days left

Contact: Kathrein Kwaik (Chatrine Qwaider)

(This lab was designed by Kathrein Kwaik (Chatrine Qwaider))

Introduction

In this assignment, we deal with a Language Identification task. The task is to read a corpus divided into 6 languages, build a language model for every language in the corpus, and test your language models. You may use the NLTK library.

We will build n-grams (bigrams and trigrams) language models from training text data. A bigram is a contiguous 2-words sequence in a sentence. For example, a sentence “Paris is the capital of France” will have the following bigrams:

[(Paris,is),(is,the),(the,capital),(capital,of),(of,France)]

Download the corpus

Both the training and test corpus can be downloaded from (eduserv.flov.gu.se:/usr/local/courses /nlp/wortschatz-leipzig-corpus) . You can use you university credentials to access the server using `ssh`. The original source of the text corpus is the wortschatz leipzig corpora. The six languages chosen are German, English, Spanish, French, Italian and Dutch.

use this command on the terminal to copy the folder from the server to your local home folder

```
scp -r guAccount@eduserv.flov.gu.se:/usr/local/courses/nlp/wortschatz-leipzig-corpus ./
```

The training corpus is assembled with examples from each of the languages we wish to identify, then we use the training information to identify what language a set of test sentences is in. Every training

file contains **30K** sentences where the corresponding test file contains **10K** sentences. **Remember to put the train and test folders in the same folder with your python file.**

Note: the text files should be read in Unicode format (UTF-8 encoding).

```
With open(file_path,"r","utf-8") as train_file:

"""
# do some operations here
"""
```

Pre-process the corpus

Pre-processing is required in order to standardize the text files. The pre-processing aims at enhancing the accuracy of the identification task. You have to implement the following steps before building the language model:

1. Convert all the text to lower case
2. Removed all the digits, punctuation marks and special characters like (!*%&?€#@£$∞§|[](){})
3. Implement a pre-processing function that takes a file as an argument and returns the pre-processed version. You can use a regex which is easier like the following

```
def pre-process(file_path):
"""
      Read the train file
      For every line/sentence
        # ……
        #line = re.sub(r"\d+", "", line) # remove digits
        # ……..
      Return the pre-processed text.
"""
```

Build the n-grams models

Every sentence of each language is tokenized (in words). You have to generate bigrams and trigrams and count their occurrences. Sort the bigrams list in descending order according to their frequencies and use the most frequent ones (from the training corpus) to produce the bigram language model.

```
French_bigram_model = sorted(n-grams, key=lambda item:
item[1],reverse=True)
```

Note: you have to compute the frequencies of each n-gram, dividing their occurrences by the total number of n-grams.

```
def train_language_mode(file_path, lang_name)
"""
#for every language
         #read the data file
         # build the n-grams
          n-grams= ngrams(sentence.split(), number_of_grams) # to build a
n-grams
         # for every n-gram in n-grams
         #count the occurrences and compute the frequency/probability
"""
```

Test the n-grams models

Again, generate a bigram list from the sentences in the test files for every language and compute the frequencies for these n-grams.

Compute the probability of each sentence belonging to every language. You will fetch the n-gram probability from the previous train_gram_model where we computed all the frequencies and probabilities.

This is an example: if we notice that (he is) as a bigram occurs 5 032 times and the total number of bigrams were 387 276 then the probability of this bigram will be 5 032/387 276 = 0,012.

To compute the total sentence probability, we multiply each of the n-grams probabilities. Notice that when we multiply a lot of small probabilities our final result gets very small; for this reason, we usually use log probabilities instead and add them (using the following identity: log(a*b) = log(a) + log(b)). Previously we calculated that (he is) has a probability of 0,012 in English, this corresponds to a log probability of log(0.0074) = -1.9.

A language with maximal probability is chosen as the language of the test sentence. This sort of statistical model always works better with more data, so longer sentences will usually be more reliably classified than short sentences.

```
def test_language_model(file_path, lang_name, n_grams_models)
"""
     #read the test file
     # for every sentence
           # build the n-gram models
           # compute the probabilities from the training model (for every
language)
           # assign the max-probability to the guessed class/language.
    # compute the accuracy of the model on every test language. ( by
comparing the guessed language with the file language name)

"""
```

Print the statistics of the corpus

For every language print the name of the language, number of sentences in both the train and test files, number of bigrams and number of trigrams as mentioned below.

run your code from the main function (#you can do changes in the main part as you code requiered)

```
if __name__ == "__main__":
lang_name = ["french","english","german","italian","dutch","spanish"]
lang_path = ["french/french.txt","english/english.txt","dutch
/dutch.txt","italian/italian.txt","germany/germany.txt","spanish
/spanish.txt"]

for i,file_name in enumerate(lang_path):
    train_language_model("train/"+file_name, lang_name[i])

for i,file_name in enumerate(lang_path):
    test_language_model("test/"+file_name,lang_name[i], n_grams_models)
```

Try to find a solution for n-grams (in the test sentences) that do not occur in the training model.

Submitting your answers

Please submit the lab as a single Python file with a name following the pattern lab2-surname.py. For example, if I were to submit, I would use lab2-kwaik.py.

Submit the file in GUL. The file should run from the command line without arguments, and print out all answers on the terminal, for example:

$ python lab2-kwaik.py
"""Assignment 2: Language Identification (deadline: 2018-12-12)
Name 1: NAME
"""
Corpus Statistics

(...)

Language Name: French

Training sentence: 30000

Testing Sentence: 10000

Number of bi-grams : 554928

Number of tri-grams : 524928

Testing Language model

---

1. **Bi-grams model**

Language Name: French

Accuracy: ---

Correctly classified: ---

Mis-classified: ---

(...)

Total Accuracy : .....

The code must be well documented, and all functions (no exceptions!) must have docstrings. All computations must be done in functions, the only things that are allowed on the top-level are, in this order:

1. module imports
2. definitions of constants
3. function and/or class definitions
4. a final run-time clause if __name__ == '__main__'

Final Notes: Code that uses sklearn or other libraries different from NLTK will not be accepted. Also note that nltk.model and nltk.lm both support bigrams at the character level and not at the word-level as we asked you to build.

Hemanth Kumar Battula , 12 Dec 2018 21:33

File name: lab2-battula.py (8,6 KB)

Status set to: To be marked

Chatrine Qwaider , 3 Jan 2019 11:49

Status set to: Completed

Comment: Well done. Thank you very much for all organising code.
you have a good programming skills.
wish you the best and happy new year.

Chatrine Qwaider , 18 Jan 2019 09:43

Grade set to: VG

Chatrine Qwaider , 29 Jan 2019 12:51

Mark set to: 7