Assignment #2: N-Gram Language Models Applied to the Task of Language Identification

Issued: 09/20/2022
Due: 10/04/2022 by 11:59PM Central Time

Total points: 100

The goal of this assignment is to give you hands on experience with the process of:

- Data pre-processing - know your data!
- Handling of n-grams
- Smoothing techniques

Thus, you are asked to implement and experiment with some form of statistical language modeling, and to write a short report about your experiences and findings. As you know, n-gram language models can be used to either generate text or to assess which sentence out of several (for example, generated by OCR, machine translation, etc.) is mostly likely. The parameters of this model will also be estimated from data.

That said, your specific task for this assignment is *Language Identification*. Language ID is the problem area of taking as input a text in an unknown language and determining what language it is written in. N-gram models are very effective solutions for this task.

For training, use the English, French, and Italian texts made available (see the src/Data/Input/ folder). For test, use the file LangId.test provided in the src/Data/Validation/ folder. For each of the following questions, the output of your program has to contain a list of space-delimited [line_id] [language] pairs, starting with line 1. For instance,

```
1 English
2 Italian
...
```

1) **<u>Question Set #1</u>** [40 points].
Implement a letter bigram model, which learns letter bigram probabilities from the training data. Thus, a separate bigram model has to be learned for each language. Then apply the models to determine the most likely language for each sentence in the test file (that is, determine the probability associated with each sentence in the test file, using each of the three language models).

Design decisions:

a) What do you consider a token for this task and why (i.e., do you include punctuation marks as well?)? What kind of preprocessing steps, if any, do you need to apply before you feed the data into your language model?

b) What technique do you decide to use for out of vocabulary (OOV) words and why?

c) Can the letter bigram model be implemented without any kind of smoothing? If not, use add-one smoothing. Is this kind of smoothing appropriate or do you need better algorithms? Why (not)?

Compare your output file with the solution file (labels.sol found in the src/Data/Validation/ folder). How many times was your program correct?

Save the program as src/Code/letterLangId.ipynb and save the output as src/Data/Output/letterLangId.out.

2) **Question Set #2** [30 points].
Implement a word bigram model, which learns word bigram probabilities from the training data. Again, a separate model will be learned for each language.

Design decisions:

a) What do you consider as a word for this task and why (i.e., only alpha-numeric characters, or do you want to use punctuation marks as well as valid word tokens)? What kind of preprocessing steps, if any, do you need to apply before you feed the data into your language model?

b) What technique do you decide to use for out of vocabulary (OOV) words and why?

c) Can the word bigram model be implemented without any kind of smoothing? If not, try add-one smoothing. Is this kind of smoothing appropriate or do you need better algorithms? Why (not)?

Apply the models to determine the language for each sentence in the test file.
Compare your output file with the solution file provided in the src/Data/Validation/ folder (labels.sol). How many times was your program correct?

Save the program as src/Code/wordLangId.ipynb and save the output as src/Data/Output/wordLangId.out.

3) **Question Set #3** [30 points].
Same as Question#2, point c), but replace the add-one smoothing with Good-Turing smoothing. What do you do when the number of words seen once are unreliable? What strategy do you use to smooth unseen words?

Save the program as src/Code/wordLangId2.ipynb and save the output as src/Data/Output/wordLangId2.out.

Which of the language models at Question Sets #1, #2, and #3 is the best? Comment on advantages and disadvantages of these language models on the task (be as detailed as possible based on your observations).

Deliverables:

- provide a README.md file in your src/ folder. Include a detailed note (i.e., one paragraph) about the functionality of each of the programs, and complete instructions on how to run them; make sure you include your name in each program and in the README.md file; make sure all your programs run correctly. The README.md file should indicate the name of the program, a short description of the problem solved and a short description on how to run your code.
- Provide an answer.pdf file where you should include answers to all the questions above with special focus on design decisions.
- Using GitHub add, commit, and push your source code and output files, the README.md file and the answer.pdf file. You must comment your code accordingly (i.e., following good coding practice guidelines). 5 points will be deducted if any of these deliverable files is missing.

NOTE:

- you are NOT allowed to use any *language model toolkits or libraries* / packages (including the ones provided with NLTK, spaCy, CoreNLP, Gensim, Pattern, PyNLPL, TextBlob, etc.) for this assignment: use pure Python functions of your own creation[1]. You must implement the models yourself. It is ok to discuss the design decisions among yourselves in person and on Canvas, but the design justifications and the code must be yours, alone (i.e., this is an individual assignment)!
- Since the purpose of this assignment is to learn and experiment with n-gram related issues, you are allowed to use your test data for development (i.e., as a dev dataset: meaning, you can test various models/implementations and then go back to retrain the n-gram so you can achieve better results on the dev set).

[1] For a list of Python3's built-in modules, please see this link. You are encouraged to use any version of Python3 above version 3.6 as older versions are no longer being actively supported.