

Exercise 4
N-Grams and Naïve Bayes

Part 1A: Text generation using N-grams

Take a look at the file `hw4-ngram-templatel.py` (or `hw4-ngram-templatel.ipynb`). The documentation at the beginning of the code has links to two websites. They provide information about the NLTK Language Model package, and an example of how to use this package to get n-gram information. You need to study these pages, observe the similarities between the pages and the code, and then do the following:

In the code:

- (a) You need to choose a corpus. The Canvas website has text files for literature from different authors and languages. Please choose one and make the changes so that the program reads this text file. (You cannot use the Sherlock example).
- (b) Write the instructions to generate a 100-word sequence out of an n-gram. Make sure that what you print on the screen looks like human-written text, not like a vector. (You'll find the code for generating sequences in the documentation of `nltk.lm`).
- (c) Write the instructions to get the probability and the counts of a unigram (e.g. "Sherlock"), a bigram (e.g. "Sherlock Holmes") and a trigram (e.g. "Sherlock Holmes said"). You can choose the specific words; they will depend on the language and author you choose. Please get the probability numbers from the trigram model.

You need to use your code to generate the following sequences, which you will then copy-paste into a Word/LibreOffice document:

- (a) A 100-word sequence using a unigram model.
- (b) A 100-word sequence using a bigram model.
- (c) A 100-word sequence using a trigram model.
- (d) A 100-word sequence using a four-gram model.
- (e) Write a one~two sentence description of how the model is performing. Which one is performing better? What makes the output "bad"?

If you generate a string in a language other than English or Spanish, please also include a translation into English. You can translate it yourself, or you can use Google Translate. If you want to use a dataset of your own (for example, a different author that you like), you can do so. Make sure you collect at least a thousand words of their writing in a text file, and then save it with UTF-8 encoding.

The output should look similar to the screen capture below. (The formatting doesn't need to be the same, but the things it shows: the counts, the probabilities, the generated text, should be present in your output).

```
Anaconda Powershell Prompt (anaconda3)
(base) PS C:\pythonpandas> python .\hw4_ngram_generation_solution.py
=== Generated text: unigram ===
it the so, to said fresh are a that ! miss woman dr. a is the it observe horror-stricken and cried you thank lantern of and pointe
d rich it emotions. merely i ! and of and understand -- dr. her of bride essential where into wrist quite knock effect. bit in the
that be which have for was end leave sofa `` and i the `` to material the it i. be assistant he, you, too case loose call the fro
m. our my it europe to of the said it pips ca fortunes.

=== Generated text: bigrams ===
no truth, gentlemen who it was inside the effect a glitter. let us, but he was possible ; an electric blue carbuncle ! `` `` you c
an never better close to the locality appeared to see the police authorities. `` is really, and i feel that i shall see, when, and
into saxe-coburg square fountain ? `` i am sure that her flight. `` `` this evening i say that. up, glancing into the brim for fr
om the effects and hurled it is terror when i observe. after we

=== Generated text: trigrams ===
photograph ? `` for answer holmes clapped his hands together in the police-court, until the last generation. i can not tell you th
at i shall be true to him to step up to the old ancestral house at hatherley about three in the simple faith of our lives. i walke
d round it to the words i regretted the impulse which caused him to pay short visits at this, we must make the easy courtesy for w
hich this man from which i could feel its hard, and i on you. `` `` to tell you

=== Generated text: fourgrams ===
were among the heads of their profession. this man strikes even deeper, but i did n't know what to do. `` `` 'no, i can not underst
and, `` said holmes. `` you are hungry, `` i remarked. `` well, i never ! `` said i. `` there is something distinctly novel about
some of the general public were present, then ? `` `` i have just been there, and the evident confusion which the cripple showed,
made the inspector realise that the matter should be pushed as

=== Counts ===
UnigramCount('sherlock'):          97
BigramCount('sherlock holmes'):     97
TrigramCount('sherlock holmes was'): 9

=== Probabilities ===
UnigramProb('sherlock'):             0.0007715681127603048
BigramProb('sherlock holmes'):        1.0
TrigramProb('sherlock holmes was'):  0.09278350515463918
```

Part 1B: Perplexity

The file `hw4-ngram-template2.py` (also `hw4-ngram-template2.ipynb`) has a bigram model, and instructions on how to calculate the perplexity of a sentence. It splits the sentence in bigrams, and then calculates the (MLE) probability of each bigram, and the perplexity of the sentence.

```
Anaconda Powershell Prompt (anaconda3)
(base) PS C:\pythonpandas> python .\hw4-ngram-template2.py
MLE Estimates: [(['had', ('marriage')), 0.038461538461538464], ([('drifted', ('had')), 0.0024009603841536613], ([('us', ('drifted')), 0.5], ([('away', ('us')), 0.005494505494505495])
MLE Estimates: [(['should', ('i')), 0.03278145695364238], ([('succeed', ('should')), 0.0047169811320754715], ([('in', ('succeed')), 0.75], ([('horror', ('in')), 0.0005685048322910744])
MLE Estimates: [(['want', ('i')), 0.002317880794701987], ([('to', ('want')), 0.32142857142857145], ([('each', ('to')), 0.0014897579143389199], ([('cheese', ('each')), 0.0])

PP(marriage had drifted us away):44.55766606167452
PP(i should succeed in horror):62.40624422167017
PP(i want to each cheese):inf
```

You need to study the code and replace the three sentences in the template for 3 sentences in the language of your choice. Make them like this:

- Use a five-word sequence that DOES appear in the corpus.
- Use a five-word sequence that appears in your bigram/trigram model from part 1A.
- Use a four-word sequence with at least one word that does not appear in your n-gram model.

Which sentence has the lowest perplexity? Which sentence has the highest? Why?

Part 2: Spell check

The site <http://norvig.com/spell-correct.html> contains a basic probabilistic spell checker. Study the code and the website, and do the following:

- You need to change the example on the website so that it corrects Cook Islands Māori spelling. Change the program so that it takes its input from `cim-sentences.txt`.
- Modify the main function so that it listens to user input, and then finds the spelling mistakes in the input. You might need to turn the input string to lowercase letters for it to be compatible with the `cim-sentences.txt` file. You also need to control for punctuation (., !)
- You need to offer suggestions for the words that are misspelled. The output should look like the screen capture below.

The code can be tested using the following sentences:

Wrong: Kia orana kotoo mai i Rarotoga!

Correct: Kia orana kotou mai i Rarotonga!

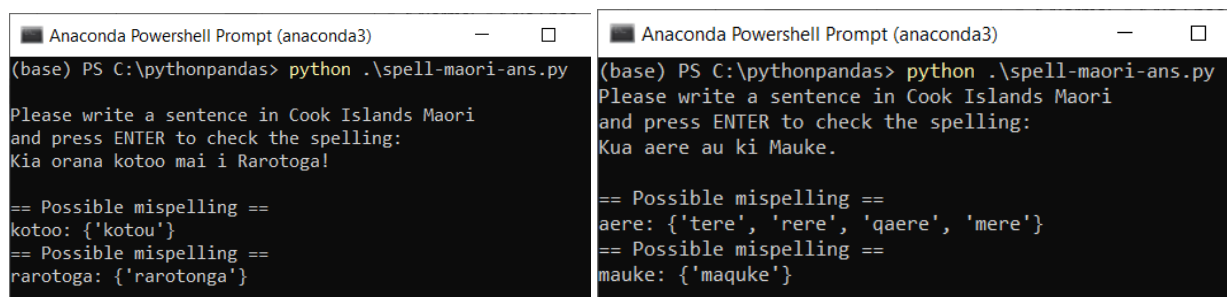
Kia orana kotou mai i Rarotonga
TAM.OPTATIVE health you.all ABL in Rarotonga¹
“Hello from Rarotonga!”

Wrong: Kua aere au ki Mauke.

Correct: Kua qaere au ki Maquke.

Kua qaere au ki Maquke
TAM.PST go I to Ma'uke
“I went to Ma'uke”.²

The output should look similar to the screen capture below. The format doesn't need to be the same, but the things it generates (a prompt to request input, the wrong words, the possible corrections), should be present in your output.



```
(base) PS C:\pythonpandas> python .\spell-maori-ans.py
Please write a sentence in Cook Islands Maori
and press ENTER to check the spelling:
Kia orana kotoo mai i Rarotoga!

== Possible misspelling ==
kotoo: {'kotou'}
== Possible misspelling ==
rarotoga: {'rarotonga'}

(base) PS C:\pythonpandas> python .\spell-maori-ans.py
Please write a sentence in Cook Islands Maori
and press ENTER to check the spelling:
Kua aere au ki Mauke.

== Possible misspelling ==
aere: {'tere', 'rere', 'qaere', 'mere'}
== Possible misspelling ==
mauke: {'maquke'}
```

¹ The glosses are given in [Leipzig Glossing](#). The correct form doesn't have many “spaces” between the words. The spaces are there to show the meaning of the morphemes in the second line. In linguistics this style of writing is called “glossing”.

² In this dataset, the ‘q’ character represents the glotal stop /ʔ/. The standard orthography uses a *salttillo* ' to represent this sound. This character is similar to an apostrophe and to the Hawaiian ‘okina.

Part 3: Naïve Bayes and Sentiment Analysis

- (1) Study the code template (`nb-sentiment-template.py`). Also, open the input files (`amazon-pos.txt`, `amazon-neg.txt`, `google-pos.txt`, `google-neg.txt`) to look at their structure.
- (2) Encapsulate the code that makes the Naïve Bayes and put it into a function. The definition of this function should look like this:

```
def runNBTest(filenamePos, filenameNeg, cutoff, numFeats):
```

This means that the function is called `runNBTest`. It takes four variables as input:

- (a) `filenamePos`: the file that has the positive reviews
- (b) `filenameNeg`: the file that has the negative reviews
- (c) `cutoff`: the percentage of reviews that should be used for the training set (e.g. 0.8,0.7,0.9)
- (d) `numFeats`: the number of "most informative features" that should be presented at the end of the analysis

You need to make a few modifications to the code so you can account for the variables `cutoff` and `numFeats`.

- (3) Run this function twice. You need to run it as follows:

```
runNBTest("amazon-pos.txt", "amazon-neg.txt", 0.8, 25)
runNBTest("google-pos.txt", "google-neg.txt", 0.8, 25)
```

- (4) Report the results of the classification in a Word/LibreOffice/PDF document. What was the accuracy, precision and recall of the program?
- (5) Report the results of the "most informative features" in a Word/LibreOffice/PDF document. You need to describe what is positive and what is negative about working at Amazon. You also need to describe what is positive and what is negative about working at Google. Finally, in addition to your written document, please submit your Python code and a JPEG/PNG file with a screen capture showing the results from your program.
- (6) (You don't need to modify the input text, but there are many ways in which these results can be improved. For example, you can remove the stopwords. You can also stem the words. Again, you don't need to do this, but if you're curious, I encourage you to examine how this changes the results, in general for the better).

Your output should look something like this:

```
ec2-user@ip-      :~/data
(base) [ec2-user@ip-      data]$ python nb-sentiment-ans.py
=== AMAZON ===
train on 800 instances, test on 200 instances
accuracy:      0.89
pos precision: 0.90625
pos recall:    0.87
neg precision: 0.875
neg recall:    0.91
neg F-measure: 0.892156862745098
pos F-measure: 0.8877551020408163
Most Informative Features
= True          pos : neg = 40.3 : 1.0
= True          pos : neg = 13.0 : 1.0
= True          pos : neg = 12.3 : 1.0
= True          pos : neg = 11.7 : 1.0
= True          pos : neg = 11.0 : 1.0
= True          pos : neg = 9.8 : 1.0
= True          pos : neg = 9.7 : 1.0
= True          pos : neg = 9.0 : 1.0
= True          neg : pos = 8.8 : 1.0
= True          neg : pos = 7.8 : 1.0
= True          neg : pos = 7.7 : 1.0
= True          pos : neg = 7.7 : 1.0
= True          neg : pos = 7.7 : 1.0
= True          neg : pos = 7.4 : 1.0
= True          pos : neg = 7.0 : 1.0
= True          pos : neg = 7.0 : 1.0
= True          pos : neg = 7.0 : 1.0
= True          pos : neg = 7.0 : 1.0
= True          pos : neg = 7.0 : 1.0
= True          neg : pos = 7.0 : 1.0
= True          neg : pos = 6.6 : 1.0
= True          pos : neg = 6.3 : 1.0
= True          neg : pos = 6.3 : 1.0
= True          neg : pos = 6.3 : 1.0

=== GOOGLE ===
train on 800 instances, test on 200 instances
accuracy:      0.885
pos precision: 0.9230769230769231
pos recall:    0.84
neg precision: 0.8532110091743119
neg recall:    0.93
neg F-measure: 0.8899521531100479
pos F-measure: 0.8795811518324608
Most Informative Features
= True          pos : neg = 29.8 : 1.0
= True          pos : neg = 25.4 : 1.0
= True          pos : neg = 21.0 : 1.0
= True          pos : neg = 17.7 : 1.0
= True          neg : pos = 15.7 : 1.0
= True          pos : neg = 15.0 : 1.0
= True          neg : pos = 14.2 : 1.0
= True          neg : pos = 13.7 : 1.0
= True          pos : neg = 13.0 : 1.0
= True          pos : neg = 12.6 : 1.0
= True          neg : pos = 12.3 : 1.0
= True          neg : pos = 12.3 : 1.0
= True          pos : neg = 10.2 : 1.0
= True          neg : pos = 9.7 : 1.0
= True          pos : neg = 9.5 : 1.0
= True          pos : neg = 9.0 : 1.0
= True          pos : neg = 9.0 : 1.0
= True          neg : pos = 8.3 : 1.0
= True          pos : neg = 8.3 : 1.0
= True          pos : neg = 8.3 : 1.0
= True          pos : neg = 8.2 : 1.0
= True          pos : neg = 7.8 : 1.0
= True          neg : pos = 7.8 : 1.0
= True          neg : pos = 7.7 : 1.0
= True          neg : pos = 7.7 : 1.0
```

Your
Amazon
features
are going
to appear
here

Your
Google
features
are going
to appear
here

What you need to submit:

Please deliver four Python scripts (one for each part) and one PDF document with your explanations for each section. When writing the code, remember to include basic documentation (your name, what the program does, its input and its output). You need to submit your code and results to Canvas before 11:59 pm EDT, Sunday, April 30th, 2023.

First part: N-grams (35%)

- 5% Computing text from {1-4}gram models
- 5% Text is presented correctly (as a readable line). Student gives an approximate translation of the sentence so the TA can grade it.
- 5% Calculate counts (code correct? present in output?)
- 5% Calculate probabilities (code correct? present in output?)
- 5% Explanation of n-grams in PDF/Word doc
- 2.5% Calculate MLE bigram probabilities
- 2.5% Calculate bigram perplexities
- 5% Explanation of perplexity results in PDF/Word doc

Second part: Spelling (20%)

- 4% Successfully reads in the CIM data (code is correctly modified)
- 16% Successfully points out errors (e.g. kotoo, aere, Rarotoga, Mauke)

Third Part: NB for Sentiment Analysis (35%)

- 10% Correctly split between pos/neg
- 10% Perform the correct train/test division
- 5% Make requested function
- 10% Analysis of probabilities and features

Format (10%)

- 5% PDF/Word document shows screenshots of results along with the explanations. The PDF/Word document has the name of the student and is easy to follow.
- 5% Code has basic metadata (name, description).

For qualitative questions (1.viii, 3.iv), the grading will be done like this:

- “Completely clear” 100% of their score: All of the questions are answered in a concise, clear and straightforward way. Examples are provided.
- “Mostly clear” 75% of their score: Most of the questions are answered in a concise, clear and straightforward manner. All of the questions are answered, but their explanation could be easier to understand or use more examples.
- “Somewhat clear” 50% of their score: The questions are answered in a cursory way, or without delving into the concepts. Few or no examples are present.
- “Unclear” 25% of their score: The explanations show misunderstandings of the concepts presented. Many of the questions are not answered.
- “No answer” 0% of their score. The student wrote no answer.