Exercise 5: Function Definitions and Frequency Distributions

Programming Techniques in Computational Linguistics 1

This is a non-graded exercise. We recommend that you solve it and hand it in by 30th November 2023 at 6 pm. If you submit it in time, we will give you feedback.

Remarks on submission

- Always name your submissions as follows: olatusernames_(pcl1|pfl)_exercise0X.txt/pdf/py/zip.
  - Example: jbiden_dtrump_pcl1_exercise05.zip
- Learning partnerships in pairs are mandatory. **Both students have to upload the solutions on olat!**
- Every file you upload, including Python programs, must contain the authors' name and the date.
- Write lots of comments! Not only will they help the tutors understand what you are trying to do, but they can also help you find and fix possible errors.
- Please number your solutions to the tasks exactly(!) the same as on the exercise sheet.
- The exercises solved on paper can be either handed in as scanned documents within your zip file or you can give us the paper with your answers in the tutorial.

1 List comprehensions and functions

Study the following function definition :

```
def foobar():
    x = []
    for y in "It's a beautiful day today, isn't it? :)":
        if not y.isalnum():
            x.append(y.upper())
    return x
```

a.) Describe what the code does in **natural language** by adding a one-line comment before each code line.

b.) Write a **list comprehension** that evaluates to the same value as the function call `foobar()`.

c.) The above function works only for the string `'It's a beautiful day today, isn't it? :)'`. This is unnecessarily specific, and we normally want to write functions that are more general. Write an entirely new function definition by giving a meaningful **name** to the function itself and all **variables**. Add a **function argument** that allows to process any string. Also write a **docstring** that describes the function according to Python documentation standards.

We recommend that you solve this exercise on paper. Otherwise, please submit an executable Python script. Write all descriptions as Python comments.

2 Local and global variables

1. Comment the following code by adding a line comment before each code line
2. Explain the concept of local and global scope as demonstrated in this code.
3. Describe the order of function calls and the corresponding outputs.
4. Explain what happens to the sentence variable at each stage of the program.

```
def func1():
    sentence = "You look beautiful today."
    print(sentence)

def outer_func():
    sentence = "Outer function sentence."

    def func2():
        global sentence
        sentence = "It's been very rainy lately."
        print(sentence)

    func1()
    print(sentence)
    func2()

sentence = "This is a third sentence."
outer_func()
print(sentence)
```

We recommend that you solve this exercise on paper. Otherwise, please submit an executable Python program.

3 Counting words

Jordan and Alex are working on a Python project. Their task is to write a function that calculates the sum of all even numbers in a given list and prints the sum. Alex completed the task, but Jordan is facing difficulties. To help, Alex showed Jordan the following disorganized and deindented code lines:

```
if num % 2 == 0:
sum_even = 0
return sum_even
def sum_of_evens(numbers_list):
for num in numbers_list:
print(sum_even)
sum_even += num
```

Try to complete this task in handwriting without using a Python interpreter.

a) Reorder and indent the provided lines to create a function that correctly calculates and prints the sum of all even numbers in a list

b) Add a doc-string that describes what the function does.

c) If this function is called and its result is assigned to a variable, what would be the content of this variable? Explain why.

d) Modify the function to also return the count of even numbers found in the list.

Please submit a Python program or a .pdf file containing your handwritten submission. If you do not have a scanner, we recommend using Adobe Scanner on your phone:
Download for Android or here for iOS

4 Reimplementing some functionality of the PERL script `tag-eval.perl` in Python

Learning goal: In this task, you should learn how to properly decompose a decently complex problem into small functions that have a clear role and meaning.

In ECL 1, we used the very, very old PERL script `tag-eval.perl` to compute detailed performance statistics for a self-trained part-of-speech tagger. See Appendix 5 on page 5 for the part of its output that you should reimplement. You can use this Binder Jupyter notebook from ECL1 to train and evaluate a POS tagger. If you are not taking the ECL 1 course, you can watch the following (crappy) Zoom recordings from Fall 2021 to get more context on the task:

- How to train and evaluate a POS tagger using hunpos and tag-eval.perl? Link to video 16 minutes
- Evaluation measures recall, precision, f-measure explained in terms of TP/FP/FN/TN values. Link to video 19 minutes Only the first 9 minutes are necessary. The following figure might help to understand the meaning of each row in the output:

		Truth	
		Positive	Negative
Test	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

found = TP+FP

present = TP+FN

4.1 Confusion Matrix & Tagging Performance per Individual POS Tag

Write a program that takes 2 tagged verticalized corpora as inputs from the command line (one key (the gold standard), and one test file). Then, compare the tags of the two corpora against each other and determine the precision, recall and f-measure for each tag present in the data.

See the Appendix for an example of what your textual output should look like (the table is the most important thing, feel free to improve the header). 5.1 contains some sample output, and 5.2 has some explanations for terms used in the output.

The file `task4.py` contains code snippets to help you print out the scores you calculate. You can modify them as you see fit. There is also a gold standard (`test.tts`) and an automatically tagged file (`result.tts`)

Your program should also do sanity checks whether the files contain the same tokens in the same order. The program should complain and exit if this is not the case.

In order to earn full points you should not just produce the correct output, but decompose your program into meaningful and properly docstring-commented functions. For instance, a function for computing recall from TP/FP statistics, etc.

4.2 *Challenge Task:* Writing the confusion matrix to an Excel File

Add an additional command line option to your evaluation script that specifies an `output.xlsx` file. Use `openpyxl` to create the specified file and write the computed statistics of your program in a suitable form in the spreadsheet.

5 Appendix

5.1 Example tag-eval output

```
CLASSIFICATION EVALUATION STATISTICS
**********************************

Key File: test.tts
 (with reference tags, treated as gold standard)

Test File: result.tts
 (with test tags, possibly wrong)
```

tag	present	found	wrong	missed	prec	recal	f-mea
ADJ	2215	2116	176	275	91.68	87.58	89.59
ADP	3448	3459	63	52	98.18	98.49	98.34
ADV	1141	1263	170	48	86.54	95.79	90.93
AUX	1063	1066	51	48	95.22	95.48	95.35
CCONJ	888	893	22	17	97.54	98.09	97.81
DET	3344	3446	128	26	96.29	99.22	97.73
NOUN	5318	5506	532	344	90.34	93.53	91.91
NUM	944	950	39	33	95.89	96.50	96.20
PART	154	164	15	5	90.85	96.75	93.71
PRON	1282	1190	30	122	97.48	90.48	93.85
PROPN	3640	3343	342	639	89.77	82.45	85.95
PUNCT	3791	3798	12	5	99.68	99.87	99.78
SCONJ	143	148	19	14	87.16	90.21	88.66
SYM	0	4	4	0	0.00	0.00	0.00
VERB	1911	1957	147	101	92.49	94.71	93.59
X	63	42	20	41	52.38	34.92	41.90
macr-avg	29345	29345	1770	1770	85.09	84.63	84.71

5.2 tag-eval help for the above output

- present: actual number of tokens with this tag in the key file
- found: number of tokens with this tag in the test file
- wrong: number of tokens in the test file that do not match their key file tag (FP)
- missed: number of tokens in the key file that have a different tag in the test file (FN)
- prec: precision
- recal: recall
- f-mea: f-measure
- macr-avg: macro-average: mean of tag-specific prec/recal/f-mea values. For all other columns, it is the sum.