AY: 2024-2025 | **M1-S2: Dept. of Electrical Engineering**
EXAM | **NLP** | **Teacher: A. Mhamdi**
June 2025 | **Time Limit: $1\frac{1}{2}$ h**

This document contains **8** pages numbered from **1/8** to **8/8**. As soon as it is handed over to you, make sure it is complete. The **3** tasks are independent and can be treated in the order that suits you.

The following rules apply:

❶ **A handwritten double-sided A4** sheet is permitted.

❷ **Any electronic material**, except basic calculator, is prohibited.

❸ **Mysterious or unsupported answers** will not receive full credit.

❹ **Round results** to the nearest thousandth *(i.e., third digit after the decimal point)*.

❺ **Task N°3:** Each correct answer will grant a mark with no negative scoring.

Task N°1

⌛ 25mn | (5 points)

Consider the following documents:

- **D1:** "To succeed, students must work hard and stay focused."
- **D2:** "Hard work leads to success for all students."
- **D3:** "Smart students work hard, but success also needs patience."

(a) (1 point) Preprocess the text:

- Convert to lowercase
- Remove punctuation (commas, periods)
- Ignore stopwords (*to, must, and, for, all, but, also*)

> - **D1:** "succeed students work hard stay focused"
> - **D2:** "hard work leads success students"
> - **D3:** "smart students work hard success needs patience"

(b) (1 point) List all unique words in alphabetical order

> ["focused", "hard", "leads", "needs", "patience", "smart", "stay", "students", "succeed", "success", "work"]

(c) (3 points) Create **BoW** vectors by counting word occurrences per document.

Document	focused	hard	leads	needs	patience	smart	stay	students	succeed	success	work
D1	1	1	0	0	0	0	1	1	1	0	1
D2	0	1	1	0	0	0	0	1	0	1	1
D3	0	1	0	1	1	1	0	1	0	1	1

Task N°2

⌛ 30mn | (5 points)

Consider the following collection of short documents about programming languages:

D1: "Python is easy to learn and widely used in data science."

D2: "Java is object-oriented and runs on many platforms."

D3: "Python and Java are popular programming languages."

D4: "Data science uses statistics and machine learning techniques."

(a) (1 point) Calculate the term frequency (TF) for each unique term in **D1**. Exclude common stop words like "is", "to", "in", "and", "on", "are".

After removing stop words, we have the terms: "Python", "easy", "learn", "widely", "used", "data", "science".

Total terms after removing stop words: 7

$$\text{TF}(\text{Python}, \textbf{D1}) = \frac{1}{7} \approx 0.143$$
$$\text{TF}(\text{easy}, \textbf{D1}) = \frac{1}{7} \approx 0.143$$
$$\text{TF}(\text{learn}, \textbf{D1}) = \frac{1}{7} \approx 0.143$$
$$\text{TF}(\text{widely}, \textbf{D1}) = \frac{1}{7} \approx 0.143$$
$$\text{TF}(\text{used}, \textbf{D1}) = \frac{1}{7} \approx 0.143$$
$$\text{TF}(\text{data}, \textbf{D1}) = \frac{1}{7} \approx 0.143$$
$$\text{TF}(\text{science}, \textbf{D1}) = \frac{1}{7} \approx 0.143$$

(b) (1 point) Calculate the inverse document frequency (IDF) for the terms "Python", "Java", "data", "science", and "programming".

Total number of documents $N = 4$

$$\text{IDF(Python)} = \log\left(\frac{4}{2}\right) = \log(2) \approx 0.301$$

$$\text{IDF(Java)} = \log\left(\frac{4}{2}\right) = \log(2) \approx 0.301$$

$$\text{IDF(data)} = \log\left(\frac{4}{2}\right) = \log(2) \approx 0.301$$

$$\text{IDF(science)} = \log\left(\frac{4}{2}\right) = \log(2) \approx 0.301$$

$$\text{IDF(programming)} = \log\left(\frac{4}{1}\right) = \log(4) \approx 0.602$$

(c) (1 point) Calculate the complete TF-IDF vector for **D1**.

TF-IDF vector for **D1**

$$\text{TF-IDF(Python, } \mathbf{D1}) = 0.143 \times 0.301 \approx 0.043$$

$$\text{TF-IDF(easy, } \mathbf{D1}) = 0.143 \times \log\left(\frac{4}{1}\right) = 0.143 \times 0.602 \approx 0.086$$

$$\text{TF-IDF(learn, } \mathbf{D1}) = 0.143 \times \log\left(\frac{4}{1}\right) = 0.143 \times 0.602 \approx 0.086$$

$$\text{TF-IDF(widely, } \mathbf{D1}) = 0.143 \times \log\left(\frac{4}{1}\right) = 0.143 \times 0.602 \approx 0.086$$

$$\text{TF-IDF(used, } \mathbf{D1}) = 0.143 \times \log\left(\frac{4}{1}\right) = 0.143 \times 0.602 \approx 0.086$$

$$\text{TF-IDF(data, } \mathbf{D1}) = 0.143 \times 0.301 \approx 0.043$$

$$\text{TF-IDF(science, } \mathbf{D1}) = 0.143 \times 0.301 \approx 0.043$$

(d) (2 points) Which terms have the highest TF-IDF scores in **D3**? What does this tell us about the document's focus?

After removing stop words, we have the terms: "Python", "Java", "popular", "programming", "languages"
Total terms: 5

$$\text{TF}(\text{Python}, \mathbf{D3}) = \frac{1}{5} = 0.2$$
$$\text{TF}(\text{Java}, \mathbf{D3}) = \frac{1}{5} = 0.2$$
$$\text{TF}(\text{popular}, \mathbf{D3}) = \frac{1}{5} = 0.2$$
$$\text{TF}(\text{programming}, \mathbf{D3}) = \frac{1}{5} = 0.2$$
$$\text{TF}(\text{languages}, \mathbf{D3}) = \frac{1}{5} = 0.2$$

Calculating TF-IDF

$$\text{TF-IDF}(\text{Python}, \mathbf{D3}) = 0.2 \times 0.301 \approx 0.060$$
$$\text{TF-IDF}(\text{Java}, \mathbf{D3}) = 0.2 \times 0.301 \approx 0.060$$
$$\text{TF-IDF}(\text{popular}, \mathbf{D3}) = 0.2 \times \log\left(\frac{4}{1}\right) = 0.2 \times 0.602 \approx 0.120$$
$$\text{TF-IDF}(\text{programming}, \mathbf{D3}) = 0.2 \times 0.602 \approx 0.120$$
$$\text{TF-IDF}(\text{languages}, \mathbf{D3}) = 0.2 \times \log\left(\frac{4}{1}\right) = 0.2 \times 0.602 \approx 0.120$$

The terms with the highest TF-IDF scores in **D3** are "popular", "programming", and "languages", all with scores of approximately 0.120. This suggests that the document's focus is on describing programming languages as popular, which is indeed the main point of the document.

AY: 2024-2025		**Full Name:**	....
M1-S2: Dept. of Electrical Engineering		**ID:**	....
Exam | **NLP**		**Class:**	RAIA1 ....
June 2025		**Room:**	....
Teacher: A. Mhamdi		**Time Limit:**	$1\frac{1}{2}$ h

Answer Sheet

Task N°3

35mn | (10 points)

(a) ($\frac{1}{2}$ point) What does the regex "^a...e$" match?

- ◯ Any string containing 'a' followed by three characters and then 'e'.
- ◯ Only the exact string a...e.
- √ Any 5-letter string starting with 'a' and ending with 'e'.
- ◯ Any string with 'a' and 'e' separated by exactly three characters.

(b) ($\frac{1}{2}$ point) What does "(abc|def)\1" match?

- √ Either "abc" or "def" followed by the same sequence again *(e.g., "abcabc" or "defdef")*.
- ◯ Either "abc" or "def" followed by "1".
- ◯ Any repetition of "abc" or "def".
- ◯ It is invalid regex syntax.

(c) ($\frac{1}{2}$ point) What is a primary feature of the polyglot library?

- √ Multilingual support with minimal configuration
- ◯ Advanced deep learning models
- ◯ GPU acceleration
- ◯ Transformer-based architectures

(d) ($\frac{1}{2}$ point) What is gensim primarily designed for?

- ◯ Part-of-speech tagging
- ◯ Named Entity Recognition
- ◯ Dependency parsing
- √ Topic modeling and document similarity

(e) ($\frac{1}{2}$ point) What information does a Bag of Words model typically discard?

◯ Word frequency ◯ Vocabulary size √ Word order ◯ Document length

Do not write anything here

✂- -

(f) ($^1/_2$ point) Which component of TF-IDF penalizes words that appear in many documents?

- ◯ Term Frequency (TF)
- ◯ Document Frequency (DF)
- √ Inverse Document Frequency (IDF)
- ◯ Normalization factor

(g) ($^1/_2$ point) What is the main advantage of TF-IDF over a simple Bag of Words model?

- ◯ It handles negation better
- ◯ It considers word order
- ◯ It reduces vocabulary size
- √ It weighs words based on their importance in the corpus

(h) ($^1/_2$ point) In a trigram model (*n=3*), what is used to predict the next word?

- ◯ The previous one word.
- √ The previous two words.
- ◯ The entire sentence.
- ◯ A random word from the vocabulary.

(i) ($^1/_2$ point) Which NLTK function would you use to split text into sentences?

- ◯ `sentence_split()`
- ◯ `text_to_sentences()`
- √ `sent_tokenize()`
- ◯ `word_tokenize()`

(j) ($^1/_2$ point) What is the correct way to load an English language model in spaCy?

- ◯ `spacy.load(”english”)`
- ◯ `spacy.load(”en”)`
- √ `spacy.load(”en_core_web_sm”)`
- ◯ `spacy.model(”english”)`

(k) ($^1/_2$ point) Which of the following is a key advantage of spaCy over NLTK?

- √ Superior speed and efficiency
- ◯ More extensive corpus collection
- ◯ Better support for regular expressions

Do not write anything here

◯ More customizable tokenization

(l) (1/2 point) In spaCy, what does the `.ents` property of a Doc object contain?

◯ All nouns in the text

◯ All verbs in the text

√ All entities mentioned in the text

◯ All adjectives in the text

(m) (1/2 point) Why are Transformers more efficient than RNNs for long sequences.

◯ They don't need gradient descent.

◯ They have fewer layers.

◯ They use CNN layers.

√ They can process all tokens in parallel.

(n) (1/2 point) Which of the following is a key feature of the Transformer architecture?

√ It uses a self-attention mechanism.

◯ It relies on recurrence or convolution.

◯ It cannot handle long-range dependencies.

◯ It is less efficient than RNNs.

(o) (1/2 point) Which operation is allowed on a tuple?

```
t = (1, 2, 3)
```

◯ `t[0] = 5` ◯ `t.pop()` √ `t += (4,)` ◯ `t.sort()`

(p) (1/2 point) What is the output of this code?

```
l1 = [1, 2, 3]
l2 = [4, 5, 6]
l1, l2 = l2, l1
print(l1[0], l2[1], sep=', ')
```

◯ 1, 5 ◯ 4, 5 √ 4, 2 ◯ 1, 2

(q) (1/2 point) What's the result of this expression?

```
def f(x=[]):
    x.append(1)
    return x
print(f(), end=', '); print(f(), end=', '); print(f())
```

DO NOT WRITE ANYTHING HERE

✂---

○ [], [], [] ○ [1], [1], [1] ○ [], [1], [1, 1] √ [1], [1, 1], [1, 1, 1]

(r) (½ point) What output will this produce?

```
d = {"a": 1, "b": 2}
l = ["a", "c"]
print([d.get(k, 0) for k in l])
```

○ [1, None] ○ ["a", "c"] ○ Error √ [1, 0]

(s) (½ point) What happens when this code runs?

```
class Parent:
    def __init__(self):
        print("Parent initialized")
class Child(Parent):
    def __init__(self):
        super().__init__()
        print("Child initialized")
c = Child()
```


○ Prints "Child initialized"

√ Prints "Parent initialized" then "Child initialized"

○ Error (must call Parent.__init__(self))

○ Prints "Parent initialized"

(t) (½ point) What does this code print?

```
class Temperature:
    def __init__(self, celsius):
        self._celsius = celsius
    @property
    def fahrenheit(self):
        return (self._celsius * 9/5) + 32
t = Temperature(25)
print(t.fahrenheit)
```

√ 77.0 ○ 25.0 ○ Error (no setter defined) ○ None of the above