Lap1

```
from google.colab import files

uploaded = files.upload()  # Mở cửa sổ chọn file

for name, data in uploaded.items():
    print(f"Đã upload: {name} ({len(data)} bytes)")
```

```
Chọn tệp en_ewt-ud-train.txt
en_ewt-ud-train.txt(text/plain) - 1013300 bytes, last modified: 7/11/2023 - 100% done
Saving en_ewt-ud-train.txt to en_ewt-ud-train.txt
Đã upload: en_ewt-ud-train.txt (1013300 bytes)
```

```
from abc import ABC,abstractmethod
from typing import List

class Tokenizer(ABC):
    @abstractmethod
    def tokenize(self,text: str)-> List[str]:
        pass
class Vectorizer(ABC):
    @abstractmethod
    def fit(self, corpus: List[str]):
        pass
    def transform(self, document: List[str]) -> List[List[int]]:
        pass
    def fit_transform(self, corpus: List[str])-> List[List[int]]:
        self.fit(corpus)
        return self.transform(corpus)
```

```
import re
from typing import List

class RegrexTokenizer(Tokenizer):
    def tokenize(self,text: str) -> List[str]:
        text = text.lower()
        tokens = re.findall(r"\w+|[^\w\s]", text)
        return tokens
```

```
import re
from typing import List

class SimpleTokenizer(Tokenizer):
    def tokenize(self,text: str) -> List[str]:
        text = text.lower()
        text = re.sub(r"([.,?!])", r" \1 ", text)
        token = text.split()
        return token
```

```
from typing import List,Dict

class CountVectorizer(Vectorizer):
    def __init__(self,tokenizer : Tokenizer):
        self.tokenizer = tokenizer
        self.vocab : Dict[str,int] = {}

    def fit(self, corpus: List[str]):
        unique_token = set()
        for doc in corpus:
            tokens = self.tokenizer.tokenize(doc)
            unique_token.update(tokens)

        self.vocab = {token: idx for idx, token in enumerate(sorted(unique_token))}

    def transform(self, document: List[str]) -> List[List[int]]:
        vectors = []
        for doc in document:
            vecto = [0] * len(self.vocab)
```

```
            tokens = self.tokenizer.tokenize(doc)
            for token in tokens:
                idx = self.vocab[token]
                vecto[idx] += 1
            vectors.append(vecto)
        return vectors

    def fit_transform(self, corpus: List[str]) -> List[List[int]]:
        self.fit(corpus)
        return self.transform(corpus)
```

```
#Task1
if __name__ == "__main__":
    tokenizer = SimpleTokenizer()
    text = "Hello , world!"
    token = tokenizer.tokenize(text)
    print(token)
```

```
['hello', ',', 'world', '!']
```

```
#Task2

if __name__ == "__main__":
    sentences= [
        "Hello, world! This is a test.",
        "NLP is fascinating... isn't it?",
        "Let's see how it handles 123 numbers and punctuation!"
    ]

    regrex = RegrexTokenizer()
    simple = SimpleTokenizer()

    for sentence in sentences:
        print(f"{sentence}",end="\n")
        print(f"regrextokenizer output: {regrex.tokenize(sentence)}", end="\n")
        print(f"simpletokenizer output: {simple.tokenize(sentence)}", end="\n")
        print("-" * 50)
```

```
Hello, world! This is a test.
regrextokenizer output: ['hello', ',', 'world', '!', 'this', 'is', 'a', 'test', '.']
simpletokenizer output: ['hello', ',', 'world', '!', 'this', 'is', 'a', 'test', '.']
--------------------------------------------------
NLP is fascinating... isn't it?
regrextokenizer output: ['nlp', 'is', 'fascinating', '.', '.', '.', 'isn', "'", 't', 'it', '?']
simpletokenizer output: ['nlp', 'is', 'fascinating', '.', '.', '.', "isn't", 'it', '?']
--------------------------------------------------
Let's see how it handles 123 numbers and punctuation!
regrextokenizer output: ['let', "'", 's', 'see', 'how', 'it', 'handles', '123', 'numbers', 'and', 'punctuation', '!']
simpletokenizer output: ["let's", 'see', 'how', 'it', 'handles', '123', 'numbers', 'and', 'punctuation', '!']
--------------------------------------------------
```

```
def load_raw_text_data(path):
    with open(path, "r", encoding="utf-8") as f:
        return f.read()
```

```
#Task3

if __name__ == "__main__":
    simple = SimpleTokenizer()
    regrex = RegrexTokenizer()

    dataset_path = "/content/en_ewt-ud-train.txt"
    rawtext = load_raw_text_data(dataset_path)
    sample_text = rawtext[:500]

    print("\n--- Tokenizing Sample Text from UD_English-EWT ---")
    print(f"Original Sample (first 100 chars): {sample_text[:100]}...\n")

    simple_tokens = simple.tokenize(sample_text)
    print(f"SimpleTokenizer Output (first 20 tokens): {simple_tokens[:20]}")

    regex_tokens = regrex.tokenize(sample_text)
```

```
    print(f"RegexTokenizer Output (first 20 tokens): {regex_tokens[:20]}")
```

```
--- Tokenizing Sample Text from UD_English-EWT ---
Original Sample (first 100 chars): Al-Zaman : American forces killed Shaikh Abdullah al-Ani, the preacher at the
mosque in the town of ...

SimpleTokenizer Output (first 20 tokens): ['al-zaman', ':', 'american', 'forces', 'killed', 'shaikh', 'abdullah', 'al-ani', ',',
RegexTokenizer Output (first 20 tokens): ['al', '-', 'zaman', ':', 'american', 'forces', 'killed', 'shaikh', 'abdullah', 'al', '
```

```
if __name__ == "__main__":
    corpus = [
        "I love NLP.",
        "I love programming.",
        "NLP is a subfield of AI."
    ]

    tokenizer = RegrexTokenizer()

    vectorizer = CountVectorizer(tokenizer)

    vectors = vectorizer.fit_transform(corpus)

    print("Vocabulary:", vectorizer.vocab)
    print("Document-Term Matrix:")
    for vec in vectors:
        print(vec)
```

```
Vocabulary: {'.': 0, 'a': 1, 'ai': 2, 'i': 3, 'is': 4, 'love': 5, 'nlp': 6, 'of': 7, 'programming': 8, 'subfield': 9}
Document-Term Matrix:
[1, 0, 0, 1, 0, 1, 1, 0, 0, 0]
[1, 0, 0, 1, 0, 1, 0, 0, 1, 0]
[1, 1, 1, 0, 1, 0, 1, 1, 0, 1]
```