**BÁO CÁO TUẦN 1**

Người làm: Trần Đức Thọ

Bài 1:

#tạo class

class Solution():

   def wordBreak(self, s, wordDict):

      self.memo = {}

      return self.solve(s,wordDict)

#hàm kiểm tra và thêm dấu cách từ

   def solve(self,s, wordDict):

      if not s:

         return ['']

      if s in self.memo:

         return self.memo[s]

      ret = []

      for i in range(1,len(s)+1):

         if s[:i] in wordDict:

            for j in self.solve(s[i:],wordDict):

               ret.append((s[:i] + " " + j).strip())

      self.memo[s] = ret

      return self.memo[s]

ob = Solution()

print(ob.wordBreak("catsanddog",["cat", "cats", "and", "sand", "dog"]))

Giải thích ý tưởng

- Chúng ta có chuỗi s và từ điển wordDict

- Chúng ta tạo 1 bản nhớ (memo) đây là kết quả chúng ta sẽ tính ra.

- Nếu chuỗi s rỗng chúng ta trả về giá trị rỗng luôn

- tạo 1 mảng rỗng(ret)

- Nếu chuỗi s là tập con có sẵn trong bản nhớ chúng ta trả về bản nhớ luôn.

- Cho i chạy từ 1 tới độ dài chuỗi s xong kiểm tra xem từ phần từ 0 tới phần tử i-1 của chuỗi có từ nào thuộc worddict không nếu có chạy 1 hàm j để chạy tiếp đệ quy chuỗi từ I tới hết.

- Thêm khoảng trắng vào giữa các từ tìm được, tiếp theo hàm strip để loại bỏ khoảng trắng 2 đầu

- Giá trị ret tính được sau khi xong bằng kết quả chúng ta cần tìm, gán ret vào memo

(phương thức em đều tham khảo trên mạng)

Bài 2

Em chưa hiểu đề bài, có vẻ như đề bài bị sai với ma trận mxn kí tự như dưới:

Calendar

Description automatically generated

Và với danh sách từ **words** = ["oath","pea","eat","rain"] em có thể tìm kiếm được 3 từ eat, oath và rain, chứ không chỉ 2 từ như output mẫu

Bài 3

+ - Khái niệm: Matplotlib là một thư viện vẽ đồ thị cho ngôn ngữ lập trình Python và phần mở rộng toán học số NumPy của nó.

- Các đồ thị vẽ ra nhằm mục đích chúng ta có cách nhìn tổng quan, có khái quát đánh giá dữ liệu để có thể đề ra giải pháp tận dụng, phân tích dữ liệu

- Một số dạng visualize data: Biểu diễn dữ liệu dưới dạng điểm, biểu đồ dạng cột, dạng hình quạt.

+ -n-gram: hiểu một cách đơn giản, phương pháp n-gram là phương pháp dự đoán từ x có khả năng xuất hiện sau từ y với xác suất là bao nhiêu dựa trên n-1 từ trước đó, các công thức xác suất dựa trên tập mẫu ban đầu (thường là rất lớn):

Ví dụ, chúng ta có tập mẫu với 5 câu:

1. He said thank you.
2. He said bye as he walked through the door.
3. He went to San Diego.
4. San Diego has nice weather.
5. It is raining in San Francisco.

Giả sử với n=2 với khả năng xuất hiện từ “you” sau từ “thank” được tích bằng thương của số lần xuất hiện từ “thank you” cho từ “thank” = 1/1= 100%

Ở đây do tập mẫu quá nhỏ nên xảy ra hiện tượng 100%

Xác suất xuất hiện từ “Diega” sau từ “San” =2/3

-Bag of Words(bow): từ tập mẫu chúng ta biểu diễn ra 1 tập các từ điển, với mỗi câu muốn được biểu diễn dưới dạng bow: chúng ta sẽ có 1 chuỗi các vector n từ mẫu, mỗi giá trị từ mẫu biểu diễn dưới dạng số lần xuất hiện từ mẫu trong câu:

Ví dụ chúng ta có 2 câu:

1. John likes to watch movies. Mary likes movies too.
2. John also likes to watch football games.

Từ 2 câu trên chúng ta có thể tạo ra 1 từ điển với 10 từ như sau:

["John", "likes", "to", "watch", "movies", "also", "football", "games", "Mary", "too"]

Bây giờ có thể biểu diễn 2 câu trên dưới dạng 2 vector

(1) [1, 2, 1, 1, 2, 0, 0, 0, 1, 1]

(2) [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]

-Tf-Idf: là công thức đánh giá sự quan trọng của một từ trong văn bản

TF (Term frequency): Tần suất xuất hiện của 1 từ trong 1 document.

TF (t, d) = (Số lần xuất hiện từ t) / (Tổng số từ)

IDF (Invert Document Frequency): Dùng để đánh giá mức độ quan trọng của 1 từ trong văn bản

IDF (t, D) = log e (Số văn bản trong tập D /Số văn bản chứa từ t trong tập D)

TF-IDF (t, d, D) = TF (t, d) \* IDF (t, D)

-Demo bag of words:

from keras.preprocessing.text import Tokenizer

sentence = ["John likes to watch movies. Mary likes movies too."]

def print\_bow(sentence: str) -> None:

    tokenizer = Tokenizer()

    tokenizer.fit\_on\_texts(sentence)

    sequences = tokenizer.texts\_to\_sequences(sentence)

    word\_index = tokenizer.word\_index

    bow = {}

    for key in word\_index:

        bow[key] = sequences[0].count(word\_index[key])

    print(f"Bag of word sentence 1:\n{bow}")

    print(f'We found {len(word\_index)} unique tokens.')

print\_bow(sentence)

Thuật toán SVM

Máy vector hỗ trợ (Support vector machine) được đề cử bởi V.Vapnik và các đồng nghiệp năm 1970s. SVM là phương pháp phân lớp tuyến tính, với mục đích xác nhận một siêu phẳng để phân tách hai lớp của dữ liệu- ví dụ: lớp các ví dụ có nhãn dương và các lớp vĩ dụ có nhãn âm. Thường thì chúng ta cần tạo một mặt diêu phẳng phân tách có lề cực đại để bài toán thêm chuẩn xác.