



TRƯỜNG ĐẠI HỌC QUỐC GIA HÀ NỘI
ĐẠI HỌC CÔNG NGHỆ



TIỂU LUẬN

MÔN: NHẬP MÔN AN TOÀN THÔNG TIN

**ĐỀ TÀI: ỨNG DỤNG CỦA MACHINE LEARNING VÀ THUẬT
TOÁN DI TRUYỀN TRONG BỂ KHOÁ SUBSTITUTION CIPHER**

Họ và tên: Nguyễn Hải Phong

Mã sinh viên: 20020065

Lớp: QH-2020-I/CQ-C-CLC



MỤC LỤC

Chương I: Giới thiệu về Substitution Cipher

1. Giới thiệu về Substitution Cipher
2. Mã hoá bằng Substitution Cipher

Chương II: Markov model And Chain Rule

1. Chain rule (probability)
2. Sử dụng N-gram để xây dựng Markov Model
3. Markov Model

Chương III: Giới thiệu về thuật toán di truyền

Chương IV: Ứng dụng của Machine Learning và thuật toán di truyền trong giải mã Substitution Cipher

1. Xây dựng thuật toán
2. Kết quả giải mã

Chương V: Source code và Tài liệu tham khảo

CHƯƠNG I: GIỚI THIỆU VỀ SUBSTITUTION CIPHER

1. Giới thiệu chung

- Substitution Cipher (hệ mật thay thế) là một phương pháp mã hóa trong đó các “unit” của bản rõ được thay thế bằng bản mã, theo một cách xác định, với sự trợ giúp của khóa. Một “unit” có thể là các chữ cái đơn lẻ, các cặp chữ cái, bộ ba chữ cái, hỗn hợp của những điều trên, v.v. Người nhận giải mã văn bản bằng cách thực hiện quá trình thay thế nghịch đảo để trích xuất thông điệp gốc.
- Tính bảo mật của hệ mật thay thế: mặc dù số lượng hoán vị rất lớn ($26! \approx 2^{88.4}$), hệ mật không thực sự mạnh và có thể bị phá vỡ bằng cách phân tích tần suất xuất hiện của các chữ cái.
- Ví dụ:

Plaintext alphabet	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Ciphertext alphabet	ZEBRASCDFGHIJKLMNOPQTUVWXY

A message

flee at once. we are discovered!

enciphers to

SIAA ZQ LKBA. VA ZOA RFPBLUAOAR!

2. Mã hoá bằng Substitution Cipher

Trong bài viết này tôi muốn trình bày 1 hướng tiếp cận đơn giản để mã hoá 1 bản rõ bằng Substitution Cipher.

```
#encrypt text based on map_Table
def encrypt(text, map_Table):
    text = text.lower()
    special_Char = re.compile('[^a-zA-Z]')
    text = special_Char.sub(" ", text)

    encoded_Text = []
    for ch in text:
        ch = map_Table.get(ch, ch) # if space then ignore
        encoded_Text.append(ch)

    return "".join(encoded_Text)

key_Table = [chr(i) for i in range(ord('a'), ord('z') + 1)]
mapped = [chr(i) for i in range(ord('a'), ord('z') + 1)]

random.shuffle(mapped)
map_Table = {key_Table[i]: mapped[i] for i in range(26)}

plain_Text = '''The Foundation is committed to complying with the laws regulating
charities and charitable donations in all 50 states of the United
States. Compliance requirements are not uniform and it takes a
considerable effort, much paperwork and many fees to meet and keep up
with these requirements. We do not solicit donations in locations
where we have not received written confirmation of compliance. To
SEND DONATIONS or determine the status of compliance for any
particular state visit'''

encoded_Text = encrypt(plain_Text, map_Table)
```

Ở đây, hàm `encrypt()` nhận 2 tham số:

- text: là bản rõ, chỉ gồm các chữ cái.
- map_Table: 1 map ánh xạ 1 chữ cái sang 1 chữ cái khác.
(được generate ở dưới)

Hàm này sẽ encrypt từng chữ cái trong bản rõ với chữ cái tương ứng của nó trong map_Table, và kết quả được đưa vào encoded_Text.

Kết quả:

- map_Table:

```
{'a': 'r', 'b': 'j', 'c': 'l', 'd': 'w', 'e': 'n', 'f': 'k',  
'g': 'm', 'h': 'i', 'i': 'g', 'j': 'c', 'k': 'v', 'l': 'b',  
'm': 'o', 'n': 'q', 'o': 'a', 'p': 'y', 'q': 'f', 'r': 'x',  
's': 's', 't': 'u', 'u': 'z', 'v': 't', 'w': 'p', 'x': 'd',  
'y': 'e', 'z': 'h'}
```

- encoded_Text (bản mã):

```
uin kazqwrugaq gs laooguunw ua laoybegqm pguí uin brps xnmzb  
rugqm lirxgugns rqw lirxgurjbn waqrugaqs gq rbb suruns ak  
uin zqgunw suruns laoybgrqln xnfzgxnonqus rxn qau zqgkaxo  
rqw gu urvns r laqsgwnxrjbn nkkaxu ozli yrynxpaxv rqw orqe  
knns ua onnu rqw vnny zy pguí uinsn xnfzgxnonqus pn wa qa  
u sabglgu waqrugaqs gq balrugaqs pinxn pn irtn qau xnlngtnw  
pxguunq laqkgxorugaq ak laoybgrqln ua snqw waqrugaqs ax wn  
unxogqn uin suruzs ak laoybgrqln kax rqe yrxuglzbrx surun tg  
sgu
```

Lưu ý: map_Table được generate 1 cách ngẫu nhiên bằng hàm [shuffle\(\)](#). Do đó nếu chạy code này có thể bạn sẽ nhận được 1 hoán vị khác.

CHƯƠNG II: MARKOV MODEL & CHAIN RULE

1. Chain Rule:

- Trong lý thuyết xác suất, Chain Rule (quy tắc chuỗi) cho phép tính toán bất kỳ phần tử nào trong phân phối chung của một tập hợp các biến ngẫu nhiên chỉ sử dụng xác suất điều kiện.

Quy tắc này rất hữu ích trong việc nghiên cứu Bayes Networks - mô tả phân phối xác suất dưới dạng xác suất có điều kiện.

- Công thức:

More than two events [[edit](#)]

For more than two events A_1, \dots, A_n the chain rule extends to the formula

$$P(A_n \cap \dots \cap A_1) = P(A_n | A_{n-1} \cap \dots \cap A_1) \cdot P(A_{n-1} \cap \dots \cap A_1)$$

which by induction may be turned into

$$P(A_n \cap \dots \cap A_1) = \prod_{k=1}^n P\left(A_k \mid \bigcap_{j=1}^{k-1} A_j\right).$$

Example [[edit](#)]

With four events ($n = 4$), the chain rule is

$$\begin{aligned} P(A_1 \cap A_2 \cap A_3 \cap A_4) &= P(A_4 | A_3 \cap A_2 \cap A_1) \cdot P(A_3 \cap A_2 \cap A_1) \\ &= P(A_4 | A_3 \cap A_2 \cap A_1) \cdot P(A_3 | A_2 \cap A_1) \cdot P(A_2 \cap A_1) \\ &= P(A_4 | A_3 \cap A_2 \cap A_1) \cdot P(A_3 | A_2 \cap A_1) \cdot P(A_2 | A_1) \cdot P(A_1) \end{aligned}$$

- Sử dụng công thức trên, ta xây dựng 1 hàm tính toán xác suất xuất hiện của 1 từ:

Giả sử $p(\text{word})$ là xác suất của từ “word”. Ta sẽ phân rã từ này ra theo công thức trên như sau:

$$\begin{aligned} p(\text{word}) &= p(d | \text{wor}) * p(r | \text{wo}) * p(o | w) * p(w) \\ &= p(d | r) * p(r | o) * p(o | w) * p(w) \end{aligned}$$

(Do trạng thái n chỉ phụ thuộc vào trạng thái n -

In the case where S is a discrete set with the [discrete sigma algebra](#) and $I = \mathbb{N}$, this can be reformulated as follows:

$$P(X_n = x_n | X_{n-1} = x_{n-1}, \dots, X_0 = x_0) = P(X_n = x_n | X_{n-1} = x_{n-1}).$$

2. Markov Model:

- Trong lý thuyết xác suất, mô hình Markov là một mô hình ngẫu nhiên mô tả 1 tập các sự kiện có thể 1 cách tuần tự khi mà xác suất xảy ra của trạng thái tiếp theo chỉ phụ thuộc vào trạng thái trước đó. (nghĩa là nó thoả mãn Markov Property).
- **Markov Assumption:** cung cấp nền tảng cho việc mô hình hoá việc ra quyết định trong các tình huống mà kết quả là một phần ngẫu nhiên và 1 phần dưới sự điều khiển của người ra quyết định. Ở đây, chúng ta sử dụng Markov Assumption để tính toán xác suất xuất hiện của 1 từ (dùng ở mục 1).

3. Sử dụng N-gram để xây dựng Markov Model:

- N-Gram là một chuỗi liên kết của N items từ một mẫu văn bản hoặc bài phát biểu. N-Gram có thể bao gồm các khối từ lớn hoặc các tập hợp âm tiết nhỏ. N-Gram được sử dụng làm cơ sở cho các mô hình N-Gram hoạt động, là công cụ trong quá trình xử lý ngôn ngữ tự nhiên như một cách dự đoán văn bản hoặc bài phát biểu sắp tới.
- Ở đây chúng ta sẽ dùng Bigram và Unigram để xây dựng Markov Model (vì theo công thức ở mục 1, ta chỉ cần tính xác suất của $p(d | r)$ và $p(w)$):

```
#create a 26x26 matrix with all values set to 1
bigram_Matrix = np.ones((26, 26))
#create a 26x1 matrix with all values set to 1
unigram_Matrix = np.ones(26)

# initializing by 1 not 0 to avoid 0 probability
# for words that didnt appear in train text
```

- Ở đây ta định nghĩa thêm 1 số function để xử lý, update N-gram Matrix:

```
def get_Index(ch):
    return ord(ch) - ord('a')

def update_Bigram_Matrix(a, b):
    bigram_Matrix[get_Index(a), get_Index(b)] += 1

def update_Unigram_Matrix(ch):
    unigram_Matrix[get_Index(ch)] += 1

# p(word) = p(d | wor) * p(r | wo) * p(o | w) * p(w)
#           = p(d | r) * p(r | o) * p(o | w) * p(w)
def update_Prob_Matrixs(words):
    words = words.split()
    for word in words:
        update_Unigram_Matrix(word[0])
        for i, ch in enumerate(word[: - 1]):
            update_Bigram_Matrix(ch, word[i + 1]) # update 2 adjoining characters
```

- Hàm `update_Bigram_Matrix` và `update_Unigram_Matrix` thao tác trên từng cặp chữ cái / 1 chữ cái.
- Hàm `update_Prob_Matrixs` cập nhật xác suất của 1 từ bằng cách tách nó ra thành Bigram và Unigram.

CHƯƠNG III: GIỚI THIỆU VỀ THUẬT TOÁN DI TRUYỀN

1. Giới thiệu chung

- Thuật toán mô phỏng quá trình tiến hoá tự thích nghi của các quần thể sinh học dựa trên học thuyết Darwin để tìm lời giải các bài toán tối ưu. Quá trình phát triển của mỗi quần thể tuân theo quy luật chọn lọc tự nhiên, tiến hoá qua các thế hệ kế tiếp nhau.
- Các hậu duệ được sinh ra từ thế hệ trước thông qua quá trình sinh sản (di truyền và biến dị) cạnh tranh tự nhiên, cá thể nào thích nghi cao hơn với môi trường thì sẽ có khả năng lớn hơn

trong tồn tại và sản sinh con cháu (cá thể nào “khỏe mạnh” sẽ sống sót và phát triển, các cá thể “yếu” sẽ bị loại bỏ).

2. Lược đồ thuật toán

```
{  
     $t \leftarrow 0$ ;  
    Khởi tạo quần thể  $P(t)$ ;  
    while (chưa kết thúc)  
    {  
         $t \leftarrow t + 1$ ;  
        Chọn lọc  $Q(t)$  từ  $P(t)$ ; // nhờ bánh xe xổ số  
        Tái tạo  $P(t)$  từ  $Q(t)$ ;  
        Đánh giá  $P(t)$  và chọn lọc cá thể tốt nhất;  
    }  
}
```

3. Các bước tiến hành

- Bước 1. Chọn cấu trúc nhiễm sắc thể biểu diễn lời giải của bài toán.
- Bước 2. Khởi tạo tập lời giải ban đầu. Việc khởi tạo là ngẫu nhiên hay có thể áp dụng một vài thuật toán heuristic
- Bước 3. Chọn hàm đánh giá mức độ tốt của lời giải (để đánh giá các cá thể trong quần thể lời giải theo độ thích nghi của chúng).
- Bước 4. Thiết kế các toán tử di truyền. Các toán tử này được thiết kế nên dựa trên nhiễm sắc thể, đặc điểm bài toán và các ràng buộc của nó.
- Bước 5: Xác định các tham số cho bài toán. Các tham số này có thể không thay đổi trong quá trình tiến hoá hoặc có thể tự điều chỉnh tham số theo thời gian.

4. Xây dựng class Decoder bằng thuật toán di truyền

```
class Genetic_Decoder:

    def __init__(self, population = 20):
        self.population = population
        self.dna_Set = []

        for _ in range(population):
            dna = [chr(i) for i in range(ord('a'), ord('z') + 1)]
            random.shuffle(dna)
            self.dna_Set.append(dna)

    def population_Score(self, encoded_Text):
        scores = []
        for i, dna in enumerate(self.dna_Set):
            current_Map = {chr(ord('a') + i): dna[i] for i in range(len(dna))}
            decoded_Text = encrypt(encoded_Text, current_Map)
            scores.append(get_Sentence_Prob(decoded_Text))

        return scores

    def remove_Weak_Entity(self, scores, keep = 5):
        #sort dna_Set based on scores
        indexs = range(len(scores))
        indexs = sorted(indexs, key = scores.__getitem__, reverse = True)

        self.dna_Set = [self.dna_Set[i] for i in indexs]
        self.dna_Set = self.dna_Set[:keep]

    def next_Gen(self, num_Children = 3):
        children = []

        #making num_Children per dna
        for dna in self.dna_Set:
            for _ in range(num_Children):
                child = dna.copy()

                #swaping two genes (mappings)
                i, j = random.randint(0, len(child) - 1), random.randint(0, len(child) - 1)
                child[i], child[j] = child[j], child[i]
                children.append(child)

        self.dna_Set.extend(children)

    def decipher(self, encoded_Text, num_Iters = 1000):
        scores = self.population_Score(encoded_Text)
        self.remove_Weak_Entity(scores) # keep 5

        for i in range(num_Iters):

            # make 3 child each so total will remain 20
            self.next_Gen(num_Children = 3)

            scores = self.population_Score(encoded_Text)
            self.remove_Weak_Entity(scores)

            if i % 100 == 0:
                print(f"Progress: {i}/{num_Iters}, Best score: {max(scores)}")

        return {chr(ord('a') + i): self.dna_Set[0][i] for i in range(26)} # [0] means get the best child
```

- `dna_Set`: cấu trúc NST biểu diễn lời giải bài toán. Ở đây số NST được khởi tạo = 20.
- `__init__()`: khởi tạo tập lời giải ban đầu.
- `population_Score()`: hàm đánh giá độ tốt của lời giải.
 - Hàm này sử dụng hàm `get_Sentence_Prob()` (sẽ được xây dựng ở phần Machine Learning) để đánh giá xác suất xuất hiện của 1 sentence.
- `remove_Weak_Entity()`: loại bỏ cá thể “yếu” dựa trên hàm đánh giá xây dựng ở trên.
- `next_Gen()`: thiết kế toán tử di truyền (toán tử đột biến).
 - Mỗi cá thể được lai tạo với chính nó và các con sinh ra được đánh giá để giữ lại cá thể tốt nhất.
- `decipher()`: hàm giải mã.
 - Hàm này nhận tham số là `encode_Text` (bản mã).
 - Hàm sẽ tính toán độ tốt của bản mã, sau đó loại bỏ các cá thể yếu (chỉ giữ lại 5 cá thể tốt nhất).
 - Tham số thứ 2 được truyền vào là `num_iters`, nghĩa là số kí tự được dùng để train, từ đó dùng để decode.
 - Giá trị trả về của hàm này là cá thể tốt nhất, nghĩa là bản rõ có giá trị hàm đánh giá cao nhất.

CHƯƠNG IV: ỨNG DỤNG CỦA MACHINE LEARNING VÀ THUẬT TOÁN DI TRUYỀN TRONG GIẢI MÃ SUBSTITUTION CIPHER

1. Xây dựng thuật toán học máy

Ở đây chúng ta xây dựng 3 hàm, theo thứ tự là:

- `get_Word_Prob(word)`: lấy xác suất của từ truyền vào sẽ xuất hiện trong bản rõ.

```
def get_Word_Prob(word):  
    logProb = 0  
  
    #adding unigram prob for first letter  
    logProb += np.log(unigram_Matrix[get_Index(word[0])])  
  
    #bigram probabilities, from second letter  
    for i, ch in enumerate(word[: -1]):  
        logProb += np.log(bigram_Matrix[get_Index(ch), get_Index(word[i + 1])])  
  
    return logProb
```

- `get_Sentence_Prob(words)`: lấy xác suất của câu văn truyền vào sẽ xuất hiện trong bản rõ.

```
#returns log probability of sequence of words  
def get_Sentence_Prob(words):  
    if type(words) == str:  
        words = words.split()  
  
    logProb = 0  
  
    for word in words:  
        logProb += get_Word_Prob(word)  
  
    return logProb
```

- Ở đây ta hiểu câu văn là 1 tập các từ liên tiếp.
- Do đó ta sẽ tính xác suất của câu văn bằng cách tính tổng xác suất của các từ trong câu văn đó. (sử dụng hàm `get_Word_Prob`).
- `create_Prob_Matrix(path)`: khởi tạo ma trận xác suất để train.

```
# build prob matrixs for train text
def create_Prob_Matrix(path):
    global unigram_Matrix, bigram_Matrix
    special_Regex = re.compile('[^a-zA-Z]')

    with open(path, 'r', encoding='utf-8') as train:
        for line in train:
            line = line.strip()

            #if non empty line
            if line:
                #replace non alpha chars
                line = special_Regex.sub(" ", line)
                words = line.lower()
                update_Prob_Matrixs(words)

    # calculate probability (normalize matrixs)
    unigram_Matrix /= unigram_Matrix.sum()
    bigram_Matrix /= bigram_Matrix.sum(axis = 1, keepdims = True)
```

- Hàm này nhận tham số là đường dẫn đến file .txt dùng để train.

2. Kết quả giải mã

- Ở đây tôi sẽ lấy 1 bản rõ bất kì, sau đó áp dụng hàm `encrypt()` để mã hoá nó. Giả định rằng chúng ta không biết `map_Table`, thuật toán sẽ tính toán xác suất của từng chữ, rồi đến từng từ và cuối cùng là đến 1 câu văn hoàn chỉnh.

```

plain_Text = '''The Foundation is committed to complying with the laws regulating
charities and charitable donations in all 50 states of the United
States. Compliance requirements are not uniform and it takes a
considerable effort, much paperwork and many fees to meet and keep up
with these requirements. We do not solicit donations in locations
where we have not received written confirmation of compliance. To
SEND DONATIONS or determine the status of compliance for any
particular state visit'''

encoded_Text = encrypt(plain_Text, map_Table)

for i in range(1,4):
    path = "test_" + str(i) + ".txt"

    decoder = Genetic_Decoder(population = 20)
    key_map = decoder.decipher(encoded_Text, num_Iters = 1000) # use first 1000 chars to decode

    with open(path, "w") as f:
        decoded_Text = encrypt(encoded_Text, key_map)

        f.write(decoded_Text)

```

- Sau khi chạy thuật toán, tôi có encode_Text như sau:

```

tqs kjhodctwjo wg ajeewttsd tj ajexpvwou mwtq tqs pcmg fsu
hpctwou aqcfwtwsg cod aqcfwtcips djoctwjog wo cpp gtcts
g jk tqs howtsd gtctsg ajexpwcoas fszhwfsesotg cfs ojt h
owkjfe cod wt tcbsg c ajogwdsfcips skkjft ehaq xcxsfmjfb
cod ecov kssg tj esst cod bssx hx mwtq tqsgs fszhwfsesotg
ms dj ojt gjpwawt djoctwjog wo pjactwjog mqsfs ms qcrs o
jt fsaswrsd mfwttso ajokwfectwjo jk ajexpwcoas tj gsod d
jocwtwjog jf dstsfewos tqs gtcthg jk ajexpwcoas kjf cov xcf
twahpcf gtcts rwgwt

```

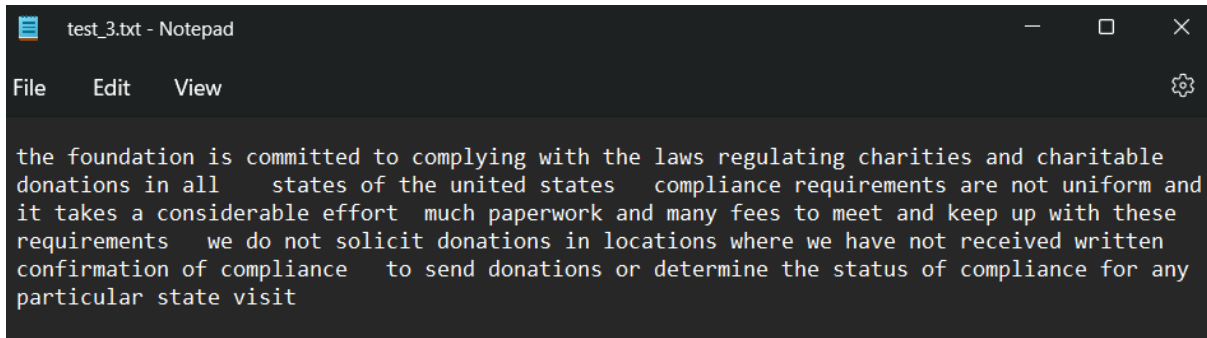
- Và kết quả của hàm đánh giá:

```

Progress: 0/1000, Best score: -1698.1312687205655
Progress: 100/1000, Best score: -1168.0119617680657
Progress: 200/1000, Best score: -1085.1026083099484
Progress: 300/1000, Best score: -993.7045644088795
Progress: 400/1000, Best score: -987.7619538426014
Progress: 500/1000, Best score: -987.7619538426014
Progress: 600/1000, Best score: -987.7619538426014
Progress: 700/1000, Best score: -987.7619538426014
Progress: 800/1000, Best score: -987.7619538426014
Progress: 900/1000, Best score: -987.7619538426014

```

- Có thể thấy, progress càng tăng thì điểm số của hàm đánh giá càng được cải thiện (cho đến 1 mức nào đó thì nó sẽ hội tụ).
- Kết quả giải mã được lưu trong file .txt được generate trong quá trình chạy:



The screenshot shows a Notepad window with the title 'test_3.txt - Notepad'. The menu bar includes 'File', 'Edit', and 'View'. The text content is as follows:

```
the foundation is committed to complying with the laws regulating charities and charitable
donations in all states of the united states compliance requirements are not uniform and
it takes a considerable effort much paperwork and many fees to meet and keep up with these
requirements we do not solicit donations in locations where we have not received written
confirmation of compliance to send donations or determine the status of compliance for any
particular state visit
```

- Có thể thấy thuật toán đã giải mã ra kết quả giống như bản rõ ban đầu.
- Thuật toán trên đã được thử nghiệm và cải tiến nhiều lần, nên hầu hết đều đưa ra kết quả đúng với thời gian ngắn.

CHƯƠNG V: SOURCE CODE & TÀI LIỆU THAM KHẢO

- Source code: [link](#)
- Tài liệu tham khảo: GoogleScholar, Github.