Monoalphabetic substitution decryption with statistical analysis attack

assignment2

제출일	2025. 10. 20	전공	컴퓨터학부
과목	정보보호론	학번	2022428418
담당교수		이름	오지우

[목차]

A. 과제의 목적

본 과제의 목적은 단일 치환 암호(monoalphabetic substitution cipher) 문장을 빈도 분석과 n-gram 통계를 사용해서 자동으로 복호화해주는 프로그램을 구현하는 것입니다. 단일 치환 암호란 평문의 각 문자를 다른 문자와 일대일로 대응시켜 암호화하는 고전 암호 방식으로, 통계적 분석으로 복호화 가능합니다.

B. 빈도 분석의 개념과 원리

문자는 문자/문자쌍/단어 통계를 낼 수 있습니다. 예를 들어 영어 문자에서는 E,T,O,A가, 문자쌍으로는 'TH'가 빈번하게 사용됩니다. 이런 통계를 기반으로 빈도 분석을 사용하면 암호문의 문자별 분포를 참고해 가장 적합한 키로 매핑해주는 방식으로 복호화할 수 있습니다.

제출 코드에서는 문자별 빈도 분석과 정확도를 높이기 위해 쿼드그램, 트라이그램, 디그램, 카이제곱, 사전 적합성, 단어 형태 패턴 등을 스코어로 합산시키는 방식을 도입했습니다.

C. 단일 치환 암호에서 빈도 분석의 문제점

단일 치환 암호에서 빈도 분석이 가지는 문제점은 짧은 암호문에서는 복호화 가능할 정도의 통계를 내기 어렵다는 점입니다. 또한 특수한 문장이거나, 분포가 균등하게 분포되어 있는 문장인 경우 또한 통계를 도출하기 어려운 경우입니다. 실제로 이번 과제의 입력값인 ciphertext1.txt 문장도 이 경우에 해당되어 빈도 분석만으로는 해석하기 어렵습니다.

A. 복호화 프로그램 설계와 코드 설명

1. 복호화 프로그램의 설계 단계

프로그램의 구조

1. 입력/전처리

   1)명령어(ciphertext1):

   python3 monoalphabetic1.py –cipherfile ciphertext1.txt –use-pangram

   2)명령어(ciphertext2):

   python3 monoalphabetic1.py –cipherfile ciphertext2.txt

   3)해당 파일들을 작업 directory로 업로드: english_quadgrams.txt, english_bigrams.txt, english_trigrams.txt, english_words.txt, ciphertext1.txt, ciphertext2.txt, monoalphabetic_1.py

2. 빈도 출력

   Counter로 각 알파벳 개수와 비율 출력

3. 스코어

   1) QuadgramScore, Trigram, Bigram

   2)디그램, 카이제곱,

   3)사전 적합성, 단어 shape 패턴 적합성

4. 탐색 알고리즘
   1) 빈도 기반의 초기 키에서 빔 서치 후 상위 후보를 유지하고 재시작되는 형태로 동작합니다.
   2) CSP 모듈
   3) 폴리싱
5. 최종 출력

   1) 평문 후보 중간 출력

   2) 최종 평문과 매칭되는 키 출력

A. 복호화 프로그램 설계와 코드 설명

2. 코드 해석

1) 키 적용

```
def substitution_key(text, key_dict):
```

2) 초기 키 생성(빈도 기반&랜덤)

```
def make_initial_key(ciphertext, rdm)
```

3) 4글자 조합 점수 계산

```
class QuadScore:
```

4) 2-3글자 조합 점수 계산

```
class NgramScore:
```

5) 보조 점수를 위한 2글자 조합 빈도수 계산

```
def bigram_bonus(text):
```

6) 반복 문자 패턴화 부분

```
def word_pattern(word):
```

7) 팬그램 점수 부분

```
def pangram_pattern_score(text_upper):
```

8) 점수 합산 부분

```
score += (WEIGHT_QUAD * q +
```

```
                WEIGHT_TRI * t +
                WEIGHT_BI * b +
                WEIGHT_DIGRAM * digram_score +
                WEIGHT_CHI * chi +
                w_dict * dict_ratio +
                w_shape * pattern_ratio +
                pang_score +
                w_penalty * weird_count)
```

9) 빔 서치

```
def beam_search(cipher_texts, quad_score, dictionary,
pattern_idx,
                        beam_size=64, neighbors_per_node=60,
max_steps=150, num_restarts=20,
              random_seed=1234, tri_score=None, bi_score=None,
              progress_interval=10, early_stop=20,
              is_short=False, use_pangram=False):
```

10) 어닐링

```
def simulated_annealing(cipher_texts, quad_score, dictionary,
pattern_idx,
                            max_iters=20000, num_restarts=10,
random_seed=7,
                               tri_score=None, bi_score=None,
is_short=False, use_pangram=False):
```

11) CSP 코드 일부

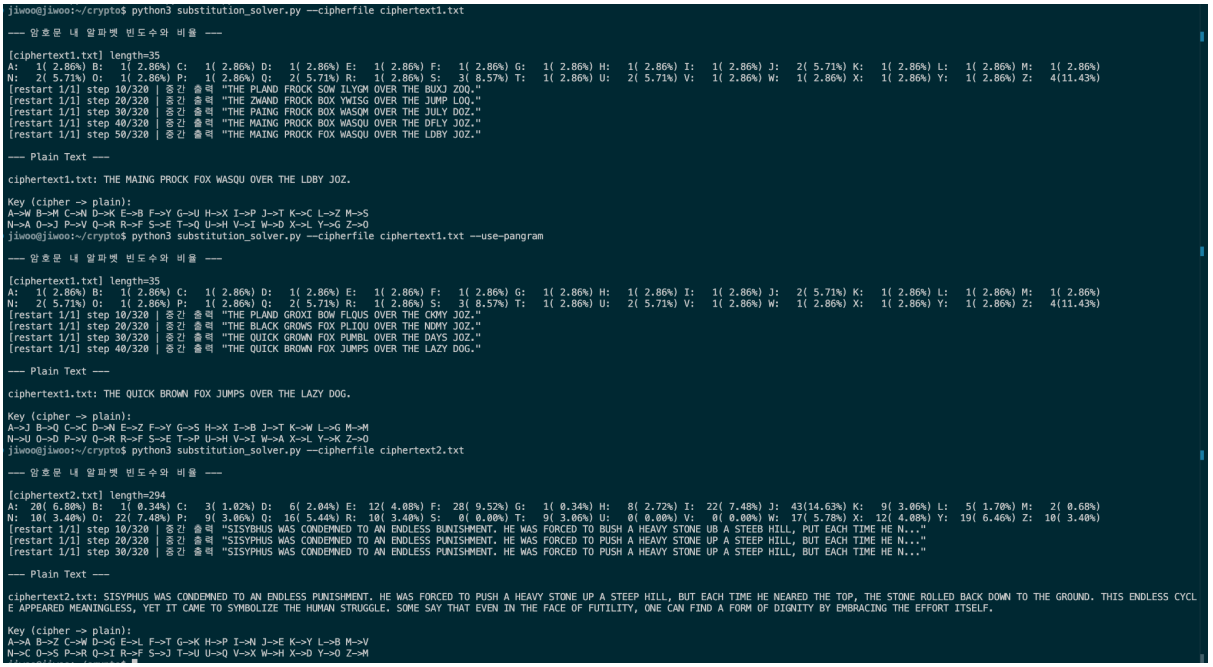
```
def invert_key(key):
    inverted = {}
    for cipher, plain in key.items():
        inverted[plain] = cipher
    return inverted


def is_consistent(current, new_mapping)
```

12) 미세 조정 부분

```
def fine_tune(cipher_texts, key, quad_score, dictionary,
pattern_idx,
                max_iters=4000, tri_score=None, bi_score=None,
is_short=False, use_pangram=False):
```

A. 복호화 프로그램 설계와 코드 설명

3. 출력 결과



- 첫 번째 출력은 pangram option을 적용하지 않은 출력이고, 각각 두 번째와 세 번째 출력은 Ciphertext1.txt와 Ciphertext2.txt를 출력한 결과입니다.

B. 문제 발생과 해결 방안

1. 문제 발생과 해결 방안

1) Ciphertext1.txt 복호화: 탐색 스텝을 최적화하고 사전/패턴 가중치를 높이고 팬그램 가중치를 적용하는 옵션을 추가하여 해결했습니다.
2) 재현성의 부족: Ciphertext1.txt의 경우 여러 번 복호화를 실행하면 랜덤성 때문에 늘 정답만을 출력하지 않는 문제가 발생했습니다. 따라서 seed를 고정하면서 이 문제를 해결했습니다.

C. 추가적인 입력이 필요한 이유

Ciphertext2.txt의 경우는 문장의 길이가 길어 추가적 옵션(팬그램 옵션)을 주지 않아도 복호화가 가능했지만, Ciphertext1.txt는 문장이 단문이고 특히 문자의 고른 분포로 복호화에서 빈도 분석을 이용하기 어려웠습니다. 고른 분포를 가진 문장의 복호화가 어려운 점은 통계적인 사용 불균형이 발생하지 않기 때문입니다. 따라서 결과적으로는 탐색 알고리즘이 잘못된 곳으로 수렴하게 되는 현상이 발생했습니다.

이를 해결하기 위해 팬그램 옵션이라는 보너스를 추가적 입력으로 받았습니다. 이 방안으로 탐색이 정답이 아닌 다른 문장으로 결정되는 것을 방지할 수 있었습니다. 특히, 여러 후보들이 모두 비슷한 ngram 점수를 나타내는 상태일 때, 일정 비율로 보상해주면서 암호문이 정답 평문에 도달하는 것이 가능했습니다.

D. 향후 개선사항 및 참고 문헌

1. 향후 개선/학습 사항

현재는 입력으로 받는 문장의 길이에 따라 조정값을 다르게 하는 코드가 작동하는 상태가 아닙니다. 문장의 길이를 탐지하면 짧은 암호문과 긴 암호문의 조정값을 별도로 설정하여 더 정확도를 높이고 싶습니다. 또한, pangram인지 아닌지를 판단하는 부분을 결정하는 부분을 수정해 별도의 옵션을 주지 않아도 자동화되어 동작하게 개선하고 싶습니다

초기에 랜덤으로 스왑해 여러 후보들을 만들고, 그 후보들로 생성하여 탐색을 시작하는 방식을 도입하여 두 방법의 결과 차이를 보고 싶습니다. 또한, 현재에는 두 파일만을 입력으로 받고 있지만 다른 여러 암호문들을 대상으로 했을 때에는 어떤 차이가 있는지 추가적으로 학습할 계획입니다.

D. 향후 개선사항 및 참고 문헌

2. 참고 문헌

1) 사전/패턴 기반 attack

   Robust Dictionary Attack of Short Simple Substitution Ciphers: Edwin Olson
   MIT Computer Science and Artificial Intelligence Laboratory Computer Engineering Cambridge, MA 02140

2) N-gram +beam+CSP 조합의 구현

   https://github.com/DarlingHang/Solver-for-Substitution-Cipher

3) 패턴 인덱싱 개념/코드

   https://inventwithpython.com/cracking/chapter17.html

4) 코드 내 rdm.shuffle(remained)의 사용 개념

   A Robust Decryption Technique Using Letter Frequency Analysis for Short Monoalphabetic Substitution Ciphers: Dayeong Kang(Kyungpook National University), Jiyeon Lee(Kyungpook National University)