CAS4107 사용자인지공학 (Prof. 이병주)

25-2 중간고사 대체 과제

Due: 2025년 10월 21일 09시 00분

개요

어떤 사용자가 스마트폰을 세로로 들고 직장 상사에게 보내는, 따라서 오타가 존재해서는 안되는 중요한 메시지를 오른손 한 손가락으로 타이핑하여 작성하고 있는 중이라고 가정하자. 사용자는 가능한 한 빠르게 메시지를 상사에게 보내기를 원한다.

이 과정에서 스마트폰 시스템은 사용자가 그동안 타이핑한 글자들을 분석하여 사용자가 입력하고자 한 단어가 무엇인지 추정하고 가장 높은 우도(likelihood)를 갖는 N개의 단어들을 사용자에게 매 글자 입력 시 마다 제안해준다 (이른바 "문구 추천", "다음 단어 제안", "자동완성" 기능). 사용자가 글자를 입력하고 시스템이 단어들을 추천하기까지 상수 시간 지연 L이 존재한다.

사용자가 특정 단어를 입력하기 위해 현재까지 입력한 글자수가 M개 일 때 시스템이 해당 시점 추천한 N개의 단어들 중 사용자가 의도한 단어가 포함되어 있을 확률 $P(N, M)$는 대략 아래 식과 같이 표현될 수 있다. 즉, 사용자가 입력한 글자수가 많아질수록 사용자가 입력하고자 한 단어가 무엇인지 추정하는 일은 시스템 입장에서 더 쉬워진다.

$$P(N, M) = 1 - \exp\left(-\frac{M \times 8.6^N}{80000}\right)$$

주어진 시스템의 키보드 레이아웃에서 한 글자(백스페이스 키 포함)를 포인팅하고 터치하는데 평균적으로 요구되는 Fitts' Law의 Index of Difficulty (ID)는 K bits라고 가정한다. 사용자의 포인팅 성능에 대한 Fitts' Law 모델의 매개변수들은 a (절편)와 b (기울기)이다. 원하는 글자를 터치하지 못하고 주변 글자를 터치하는 실수 확률은 E이다. 단, 백스페이스키를 누르는 과정에선 오류가 발생하지 않는다고 가정한다. 나아가 사용자의 선택 반응 시간은 (Choice Reaction Time) Hick's Law를 따르며 매개변수들은 c (절편)와 d (기울기)라고 가정한다. 단, 시스템으로부터 제안된 단어들 중 하나를 선택한 뒤 그것을 실제로 터치하는데 (이 과정에서의 터치 오류는 발생하지 않는다) 소요되는 운동 시간은 무시할만큼 충분히 짧다고 가정한다. Fitts' Law 및 Hick's Law는 본디 평균 포인팅 시간과 평균 반응시간을 설명하지만 이 과제에선 특별히 두 법칙이 개별 트라이얼의 포인팅 시간 및 반응시간을 예측할수 있다고 예외적으로 가정한다. 즉, 두 법칙의 종속변수가 기대값 $E[TCT]$ 혹은 $E[RT]$가 아닌 TCT와 RT 자체라고 가정하는 것이다. 사용자는 전문가로서 키보드 레이아웃을 완전히 외우고 있다고 가정하고 타이핑 과정에서 어떠한 학습효과 혹은 피로효과도 발생하지 않는다고 가정한다.

이 시나리오에서 Model Human Processor (MHP) 모형에 기반하여 사용자가 특정 단어를 타이핑하는 일련의 인지 과정을 다음과 같이 쓸 수 있다고 하자.

[Step 1]

지각 시스템 (perceptual processor)이 앞서 타이핑 완료된 단어들이 표시되어있는 화면 상태를 지각하고 이어서 인지 시스템 (cognitive processor)이 다음으로 타이핑하고자 하는 단어와 해당 단어를 이루는 글자들의 키보드 상에서의 위치를 떠올리고 손가락의 움직임 계획을 생성한다. 단어에 포함된 모든 글자들을 누를 수 있는 일련의 움직임들을 한번에 계획한다고 가정한다. Step 1에서 소요되는 시간은 MHP의 매개변수를 따른다. 단, 타이핑한 단어과 추천단어 모두를 하나의 청크(chunk)로 인식할 수 있다고 가정한다.

[Step 2]

운동 시스템이 인지 시스템으로부터 전달받은 손가락 움직임 계획을 실행하기 시작한다. 이 때는 순수 Fitts' Law만으로 소요시간이 결정된다고 가정한다. 각 글자를 타이핑하기 위한 손가락 움직임이 시작될 때, 동시에 병렬적으로 (1) 오타발생 여부 검토 및 수정 프로세스와 (2) 추천단어 검토 프로세스가 실행된다. 각 프로세스는 다음과 같다.

오타발생 여부 검토 및 수정 프로세스

새로운 글자가 타이핑 될 때마다 지각시스템은 오타가 발생되었는지 지각한다. 여기에 소요되는 시간은 MHP의 매개변수를 따르며 손가락 움직임에 드는 시간보다는 항상 작다고 가정한다.

- **만약** 오타가 발생한 것이 감지되었다면 (1) 인지 시스템은 손가락 움직임을 비롯한 모든 프로세스를 멈추기로 결정하고, 이어서 오류를 수정할 (백스페이스 키를 누를) 움직임을 계획한다. (2) 그 다음 운동시스템은 인지시스템으로부터 전달받은 오류 수정 움직임을 실행하는데, 이 때는 순수 Fitts' Law만으로 소요시간이 결정된다. (3) 오류수정 움직임이 완료되면, [Step 1]과 동일하게 지각시스템은 현재까지 입력된 글자들을 지각하고, 인지시스템은 앞으로 추가로 타이핑해야하는 글자들을 결정하고 그것들을 누를수 있는 손가락 움직임 계획을 생성한다. (4) [Step 2]의 시작점으로 돌아간다.

추천단어 검토 프로세스

새로운 글자가 타이핑 되고 추천 단어 목록이 업데이트되면, 지각시스템은 시스템이 제안한 N개의 추천 단어들을 검토하고 그 안에 목표로 한 단어가 포함되어있는지 확인한다. 여기서 확인에 소요되는 시간은 Hick's Law만으로 결정된다고 가정한다.

- **만약** 의도한 단어가 존재한다면 인지시스템은 추천단어를 누를지 아니면 기존 타이핑 움직임을 지속하여 진행할지를 결정한다. 여기서 소요되는 시간은 MHP의 매개변수를 따른다. **주의:** 인지시스템에서 인지가 완료되기 전에 다음 글자 입력에 의한 추천단어 목록 업데이트가 발생하는 경우가 존재할 수 있다.
  - **만약** 인지시스템이 추천단어를 누르기로 결정했다면 무시할만큼 짧은 소요시간으로 해당 단어를 누르고 트라이얼은 종료된다 (운동시스템 고려할 필요 없음).

모든 글자들을 오타없이 타이핑하는데 성공했다면 트라이얼은 종료된다.

과제 요구 사항

앞선 설명들에 바탕하여 5가지의 세부 과제들을 수행하라. 모든 과제들을 수행함 있어서 MHP의 매개변수들은 강의 슬라이드에 제시된 표준값을 고정적으로 사용한다: $\tau_p = 0.1$ s, $\tau_c = 0.07$ s. 각 매개변수의 정의는 강의 슬라이드를 참조하라.

[세부 과제 1] 단어제안 기능을 사용하지 않는 사용자 행동 시뮬레이션

사용자가 시스템의 단어제안 기능을 전혀 사용하지 않기로 마음먹고 있다고 가정한 상태에서 개요에 제시된 알고리즘을 탑재한, 실제 동작하는 `Python` 타이핑 행동 시뮬레이터를 구현하라. 시뮬레이터는 사용자가 타이핑하고자 하는 영문 소문자로만 이루어진 특정 단어가 주어지면 그 단어가 타이핑되는 일련의 과정을 콘솔에 프린트한다. 이 때 시뮬레이션 함수는 알고리즘의 모든 주요 매개변수들(N, L, a, b, c, d, E, K 등) 역시 입력으로 받아야한다. 완성된 시뮬레이터에 apple이라는 단어를 입력한 경우 콘솔 출력 형식은 아래와 같다 (**주의**: 아래 예시에서의 출력 값은 임의로 제시된 것으로, 실제 답안과는 차이가 있을 수 있다).

```
Time    Target  Result
0.22    a       success
0.63    p       fail
0.78    BS      success
0.91    p       success
1.11    p       success
1.34    l       success
1.83    e       success
```

위 형식에서 각 타임스탬프는 초 단위로 소수점 두번째자리까지 반올림하여 출력되어야하며 단어가 처음 주어진 시점을 $t = 0$으로 간주하여야 한다. 각 컬럼은 TAB으로 구분되어야하며 세 번째 컬럼엔 해당 키 입력이 성공했는지 실패했는지 여부를 나타낸다. 키 입력에 실패한 경우 목표로 했던 글자를 출력하면 된다. 목표로 한 글자 입력에 실패한 것이 감지되어 그것을 수정하고자 백스페이스키를 누른 이벤트는 특별히 BS라고 표현한다. 백스페이스키를 누를 때는 실수가 없다고 가정하지만 그래도 성공실패 여부를 출력하라. 제일 윗 행의 Time, Key, Result 역시 header로서 출력되어야 한다.

[세부 과제 2] 목표로 한 단어가 시스템의 제안단어 목록에 존재하는 경우 그것을 무조건 누르는 사용자 행동 시뮬레이션

사용자가 시스템의 단어 제안목록에 목표로 한 단어가 포함됨을 감지한 경우 무조건 그것을 누른다고 가정한 상태에서 페이지 2에 제시된 알고리즘을 탑재한, 실제 동작하는 Python 타이핑 행동 시뮬레이터를 구현하라. 시뮬레이터는 사용자가 타이핑하고자 하는 영문 소문자로만 이루어진 특정 단어가 주어지면 그 단어가 타이핑되는 일련의 과정을 콘솔에 프린트한다. 이 때 시뮬레이션 함수는 알고리즘의 모든 주요 매개변수들(N, L, a, b, c, d, E, K 등) 역시 입력으로 받아야한다.

완성된 시뮬레이터에 apple이라는 단어를 입력한 경우 콘솔 출력 형식은 아래와 같다 (주의: 아래 예시에서의 출력 값은 임의로 제시된 것이다).

```
Time    Target   Result
0.24    a        success
0.52    p        fail
0.73    BS       success
0.89    p        success
1.21    AC       success
```

위 형식에서 각 타임스탬프는 초 단위로 소수점 두번째자리까지 반올림하여 출력되어야하며 단어가 처음 주어진 시점을 $t = 0$으로 간주하여야 한다. 각 컬럼은 TAB으로 구분되어야하며 세번째 컬럼엔 해당 키 입력이 성공했는지 실패했는지 여부를 나타낸다. 키 입력에 실패한 경우 목표로 했던 글자를 출력하면 된다. 목표로 한 글자 입력에 실패한 것이 감지되어 그것을 수정하고자 백스페이스키를 누른 이벤트는 특별히 BS라고 표현한다. 백스페이스키를 누를 때는 실수가 없다고 가정하지만 그래도 성공실패 여부를 출력하라. 시스템이 제안한 단어 목록 중 하나를 택하여 누르고 트라이얼이 종료되는 이벤트는 AC (Auto Complete)라는 기호로 출력하라. 앞선 가정에 따라 제안단어들 중 하나를 선택하는 과정엔 실수가 발생하지 않지만 그래도 해당 이벤트의 성공실패 여부를 출력하라. 제일 윗 행의 Time, Key, Result 역시 header로서 출력되어야 한다.

[세부 과제 3] 단어 길이의 변화에 따른 단어제안 기능의 유용성 변화 탐구

이 과제는 세부과제 1과 세부과제 2에서 구현된 시뮬레이터들을 이용하여 수행한다. 직관적으로 단어 길이가 짧을수록 단어 제안 기능을 이용하기보다 직접 타이핑하는 것이 전체 타이핑 시간을 줄이는데 효과적일 것이다. 버튼 하나를 누르는데 평균 3-bits ($K = 3$) 난이도를 갖는 시스템이 지연시간 0.3초 ($L = 0.3$)로 4개의 단어를 추천하며 ($N = 4$) 사용자의 포인팅 및 반응 성능이 다음과 같은 매개변수들로 특성화될 수 있다고 가정해보자: $a = 0.1$, $b = 0.2$, $c = 0.2$, $d = 0.15$, $E = 0.04$.

단어의 길이가 변화될 때 타이핑 완료 시간의 변화를 그래프로 출력하라. 구체적으로 한 그래프에 세부과제 1 시뮬레이터의 출력(빨강, `#FF0000`)과 세부과제 2 시뮬레이터의 출력(파랑, `#0000FF`)을 함께 표시하라. 그래프의 x축을 단어의 길이, 그래프의 y축을 타이핑 완료 시간으로 그릴 것을 권장한다. 단어 길이가 몇 일 때부터 세부과제 2의 시뮬레이터가 세부과제 1의 시뮬레이터보다 더 짧은 타이핑 소요시간을 보이는가? 타이핑 과정이 랜덤성을 내포하고 있는 것을 고려하여, 같은 시뮬레이션을 1000번 반복하여 그 평균을 타이핑 완료 시간으로 계산하여라 (이후 세부과제 4, 5 역시 동일함). 이 때, 매 시뮬레이션이 같은 시드값을 사용하여 동일한 값이 샘플되지 않도록 주의한다 (최초의 시드 고정 1번만 이루어져야 한다; 자세한 내용은 후술).

[세부 과제 4] 도보 이동 중에 메세지를 보내는 사용자에 대해 단어제안 기능의 유용성 탐구

세부과제 3과 동일한 상황이지만 사용자가 도보로 이동 중에 메세지를 보내기 때문에 일반적인 시나리오보다 높은 타이핑 에러율 15%를 보인다고 가정해보자 ($E = 0.15$). 세부과제 3과 동일한 그래프를 그리고 역시 단어 길이가 몇 일 때부터 세부과제 2의 시뮬레이터가 세부과제 1의 시뮬레이터보다 더 짧은 타이핑 소요시간을 보이는지 분석하라.

[세부 과제 5] 최적 단어제안 수의 존재 분석

당신이 이 시스템을 개발한 프로젝트 리더이고 몇 개의 단어들을 제안하도록 시스템을 구현할지 이제 결정해야한다고 가정해보자. 세부과제 3과 동일한 시나리오이지만 추천단어들의 갯수 N이 변수라고 가정했을 때 세부과제 1의 시뮬레이터와 세부과제 2의 시뮬레이터 각각에 대해 최적의 단어제안 갯수이 N이 존재하는가? 존재한다면 그 값들은 얼마인가? 두 값들은 차이를 보이는가?

여기서 최적의 단어제안 갯수란 사용자의 평균 타이핑 소요시간이 최소가 되는 갯수를 의미한다. 이를 위해 N의 함수로 각 시뮬레이터의 평균 소요시간의 변화를 나타내는 하나의 그래프를 출력하라 (빨강 `#FF0000`: 세부과제 1 시뮬레이터, 파랑 `#0000FF`: 세부과제 2 시뮬레이터). 모든 영단어는 동일한 빈도로 등장한다고 가정하고 영단어의 길이는 평균 4.8, 표준편차 1.5의 가우시안 분포 ($\mathcal{N}(\mu = 4.8, \sigma = 1.5)$)를 따른다고 가정한다. 길이를 정수화하기 위해 가우시안 분포로부터 샘플링된 값은 반올림하라.

세부과제 별 배점 및 제출 포맷

Table 1: 각 세부과제 별 요구 파일 목록 및 배점

과제	제출 요구 파일	배점
세부과제 1	학번_1.py (소스코드)	25
세부과제 2	학번_2.py (소스코드)	25
세부과제 3	학번_3.py (소스코드) 학번_3.png (출력 그래프) 학번_3.pdf (질문에 대한 답)	20
세부과제 4	학번_4.py (소스코드) 학번_4.png (출력 그래프) 학번_4.pdf (질문에 대한 답)	10
세부과제 5	학번_5.py (소스코드) 학번_5.png (출력 그래프) 학번_5.pdf (질문에 대한 답)	20

위의 표에 제시된 세부 과제 별 파일을 모두 하나의 .zip 파일로 압축하여 LearnUs에 제출한다. 압축 파일의 이름은 역시 본인의 학번이다. 예를 들어, 학번이 2025123456인 홍길동 수강생의 과제 전체 압축 파일명은 `2025123456.zip`이며, 세부과제 1의 소스코드 파일명은 `2025123456_1.py`이다 (※ 압축 해제 시 모든 파일이 하나의 폴더 안에 존재하면 되며, 세부과제 별로 폴더를 분리할 필요는 없다).

무작위 샘플링 함수 구현

주어진 확률 $p \in [0, 1]$에 대하여 p의 확률로 `True`를 return하는 샘플링 함수는 다음과 같은 `numpy` (버전 무관) 기반의 샘플링 함수를 그대로 활용하여라.

```
def np_random_sampler(p):
    return np.random.rand() < p
```

모든 세부과제 구현 시 확률적 샘플링 과정은 예시처럼 `numpy.random.rand()` 함수를 활용할 것을 권장한다 (동일한 시드 설정 및 샘플링 함수 구현을 통해, 채점 시 재현성을 빠르고 원활하게 검증하기 위함으로 함수의 이름을 반드시 따라야 할 필요는 없음). NumPy의 랜덤 시드는 `None` 또는 상수로 정의되며, `argparse`를 통해 실행 단계에서 조절할 수 있도록 하여라. 랜덤 시드에 대한 argument의 명칭은 반드시 `random_seed`로 하며, default argument 값은 `None`로 한다 (아래 예시 참조, **오탈자로 인해 실행이 한번에 안될 경우, 감점 요인**).

```
import numpy as np
import argparse
parser = argparse.ArgumentParser(description="Typing simulation")
parser.add_argument("--random_seed", type=int, default=None)
args = parser.parse_args()

if args.random_seed is not None:
```

```
    NP_SEED = args.random_seed
    np.random.seed(NP_SEED)
else:
    NP_SEED = None  # true randomness

def np_random_sampler(p):
    # assert 0 <= p <= 1
    return np.random.rand() < p

(Your code ...)
```

매개변수 및 입력 대상 단어 설정 (세부과제 1, 2)

실행 단계에서 시뮬레이터의 매개변수 및 입력 대상 단어 역시 `argument`로 전달해준다. 개요에서 소개된 매개변수들을 정의하는 argument의 이름은 다음과 같다.

- 추천 단어 총 개수는 `N`로 명명하며, default 값은 4로 한다.
- 시스템이 단어를 추천하기까지의 상수 지연 시간은 `L`로 명명하며, default 값은 0.3로 한다.
- Fitts' Law의 ID는 `K`로 명명하며, default 값은 3로 한다.
- Fitts' Law 모델의 매개변수 a는 `a`로 명명하며, default 값은 0.1로 한다.
- Fitts' Law 모델의 매개변수 b는 `b`로 명명하며, default 값은 0.2로 한다.
- Hick's Law 모델의 매개변수 c는 `c`로 명명하며, default 값은 0.2로 한다.
- Hick's Law 모델의 매개변수 d는 `d`로 명명하며, default 값은 0.15로 한다.
- 글자를 잘못 터치하는 실수 확률 E는 `E`로 명명하며, default 값은 0.04로 한다.
- 입력 대상 단어는 `target_word`로 명명하며, default 값은 apple로 한다.

별도로 논리적으로 모순되는 매개변수 입력은 하지 않는다고 가정한다. **Argument 이름의 오탈자로 인해 실행이 한번에 안될 경우, 감점 요인이 될 수 있음에 유의한다.**

```
import numpy as np
import argparse
parser = argparse.ArgumentParser(description="Typing simulation")
parser.add_argument("--random_seed", type=int, default=None)
parser.add_argument("--N", type=int, default=4)
parser.add_argument("--L", type=float, default=0.3)
parser.add_argument("--K", type=float, default=3)
parser.add_argument("--a", type=float, default=0.1)
parser.add_argument("--b", type=float, default=0.2)
parser.add_argument("--c", type=float, default=0.2)
```

```
parser.add_argument("--d", type=float, default=0.15)
parser.add_argument("--E", type=float, default=0.04)
parser.add_argument("--target_word", type=str, default="apple")
args = parser.parse_args()

if args.random_seed is not None:
    NP_SEED = args.random_seed
    np.random.seed(NP_SEED)
else:
    NP_SEED = None  # true randomness
TARGET_WORD = args.target_word
(define simulator parameter variables here: N, L, K, ...)

def np_random_sampler(p):
    return np.random.rand() < p

(Your code ...)

# if __name__ == "__main__":
#    (Your typing simulator execution code snippet here)
```

다음은 터미널을 통한 실행 커맨드 예시이다.

- `python {학번}_{과제번호}.py --random_seed 2025 --target_word interaction`
- `python {학번}_{과제번호}.py --a 0.2 --E 0.1`

세부과제 3, 4, 5 구현 관련

세부과제 1, 2와 마찬가지 방법으로 실행하되, argument의 선언은 랜덤 시드만 필요하다 (즉, `python {학번}_{세부과제번호}.py --random_seed {시드}`로 실행). 소스코드 실행 시, 출력해야하는 그래프는 `matplotlib.pyplot`의 `show()` 함수를 활용하는 것을 권장한다.

```
import matplotlib.pyplot as plt
(Your code snippet here)
plt.show()
```

제출 대상인 png 파일은 수동으로 별도로 저장하여야 하며, 소스코드 실행 시 (이미지를 포함하여) 어떠한 파일이 생성 및 저장되는 코드가 포함되어 있어선 안된다 (예: `plt.savefig()`).
세부과제 1, 2는 시뮬레이터 실행 시 결과가 콘솔에 출력되는 것이 요구되지만, 세부과제 3, 4, 5에서는 비교 해석을 위해 시뮬레이션 실행 결과가 어떤 변수 형태로 저장되어야 하므로, 구현에 약간의 수정이 필요할 수 있음에 유의한다.

감점 요인

- 늦은 제출: **-100%**

- 파일명 오탈자: **-50%**
- (모든 세부과제) Argument 명칭 오탈자: **-50%**
- (세부과제 3, 4, 5) `plt.show()`를 통한 이미지 출력 이외의 별도의 파일 저장 코드 포함: **-20%**