Activity 11 - Biometrics

Code

We use the keyboard python library to record keystrokes along with the time, and construct a digraph with a python dictionary.

```
import keyboard
import json
import numpy as np
from numpy.linalg import norm

def record_digraph(digraph, times):
  print('\nType this paragraph, then press esc. on finish: (do not mind
errors)\n')
  print('the quick brown fox jumps over the lazy dog.')
  # print('the quick brown fox jumps over the lazy dog. waltz, bad nymph, for
quick jigs vex. sphinx of black quartz, judge my vow. how vexingly quick daft
zebras jump!')

  events = keyboard.record(until='escape')
  down_events = []
  for e in events:
    if e.event_type == 'down':
      down_events.append(e)

  for i in range(len(down_events)):
    if i == 0:
      continue
    charpair = down_events[i-1].name + down_events[i].name
    interval = (down_events[i].time - down_events[i-1].time) * 1000
    if charpair in digraph:
      digraph[charpair] += interval
      times[charpair] += 1
    else:
      digraph[charpair] = interval
      times[charpair] = 1

  # total time
  total_time = (down_events[-1].time - down_events[0].time) * 1000
  if '_total' in digraph:
    digraph['_total'] += total_time
    times['_total'] += 1
  else:
    digraph['_total'] = total_time
    times['_total'] = 1

  for key in digraph:
    digraph[key] /= times[key]
```

```
        print(key, digraph[key], times[key])
    print()
```

We use cosine similarity to compare two digraphs.

```
def compare_digraph_cosine(input, baseline):
  input_sorted = dict(sorted(input.items()))
  baseline_sorted = dict(sorted(baseline.items()))

  input_vect = []
  baseline_vect = []

  for key in baseline_sorted.keys():
    if key in input_sorted:
      input_vect.append(input[key])
      baseline_vect.append(baseline[key])

  input_vect = np.array(input_vect)
  baseline_vect = np.array(baseline_vect)
  return np.dot(input_vect, baseline_vect)/(norm(input_vect)*norm(baseline_vect))
```

Finally we also save the digraphs to a file so we can store the data.

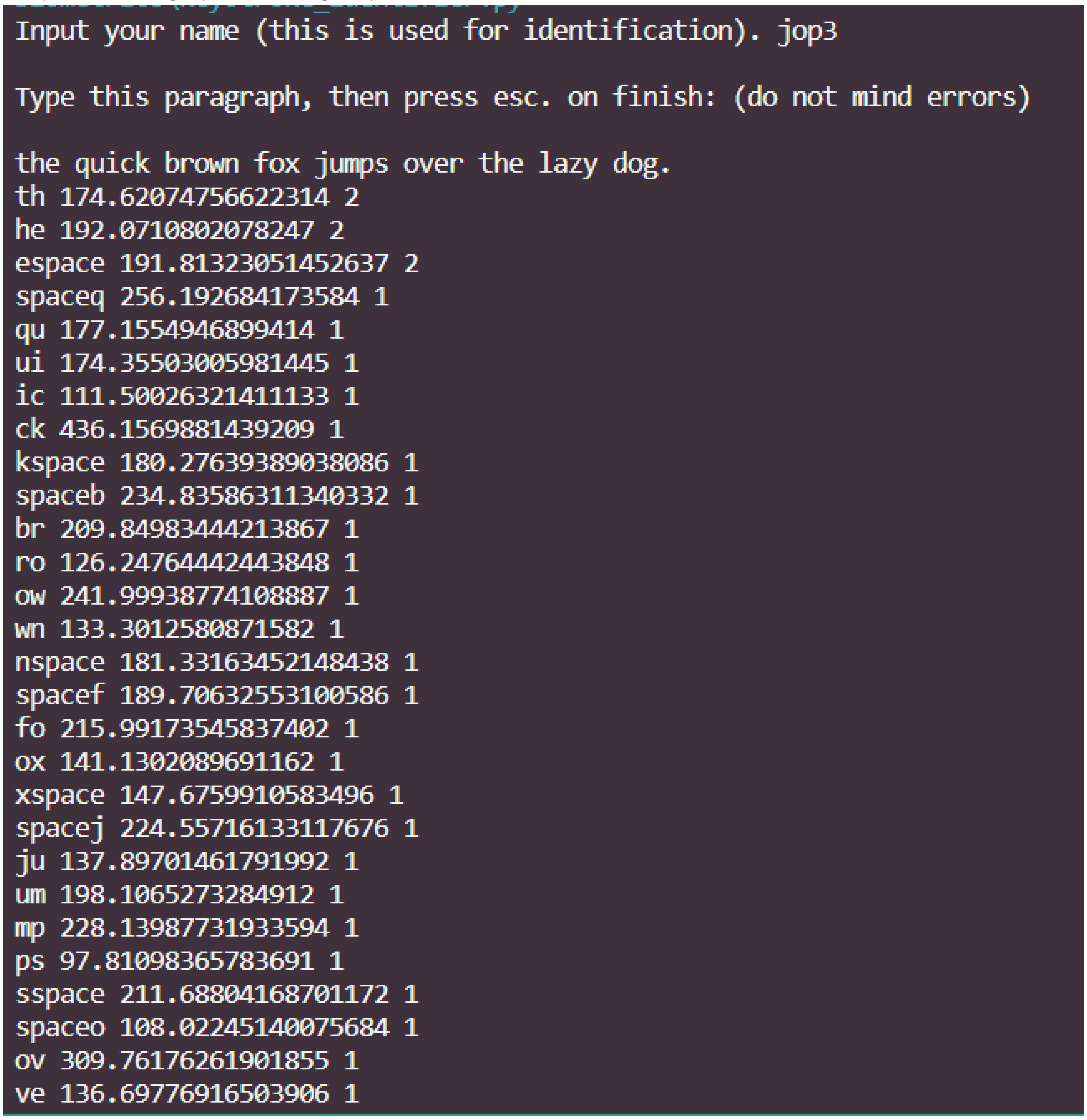
```
def save_digraph_as_json(filename, digraph):
  digraph_json = json.dumps(digraph, indent=4)
  with open(filename, "w") as file:
    file.write(digraph_json)


def read_digraph_json(filename):
  digraph = {}
  with open(filename, "r") as file:
    digraph = json.load(file)
  return digraph
```

```
# ----- Program Start -----

digraph = {}
times = {}

user = input("Input your name (this is used for identification). ")

for i in range(5):
  record_digraph(digraph, times)

save_digraph_as_json("digraph_" + user + ".json", digraph)
```

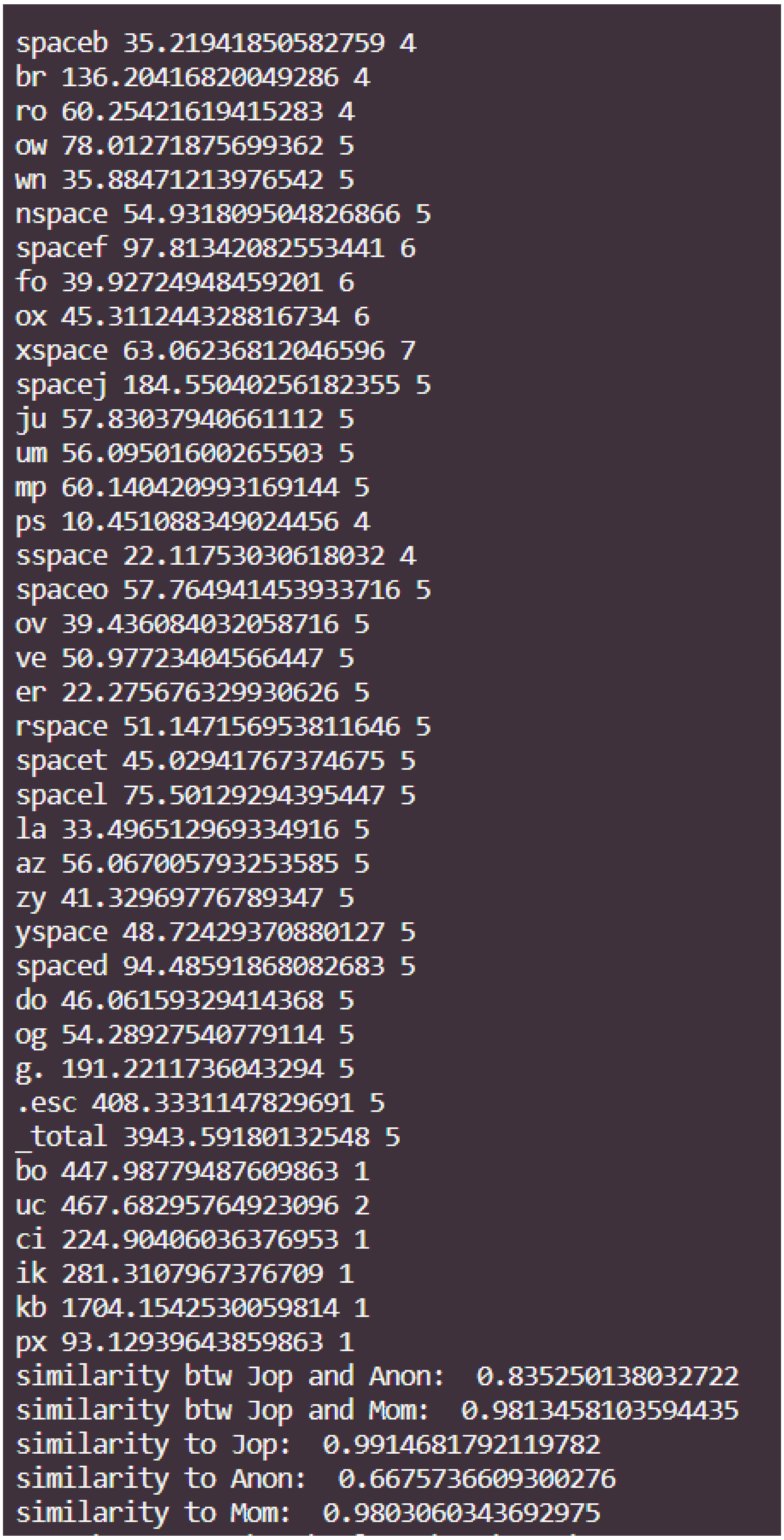
```
    for key in digraph:
        print(key, digraph[key], times[key])

    digraph_jop = read_digraph_json("digraph_jop.json")
    digraph_anon = read_digraph_json("digraph_Anon.json")
    digraph_mom = read_digraph_json("digraph_sonia.json")

    diff_jop = compare_digraph_cosine(digraph, digraph_jop)
    diff_anon = compare_digraph_cosine(digraph, digraph_anon)
    diff_mom = compare_digraph_cosine(digraph, digraph_mom)


    print("similarity btw Jop and Anon: ", compare_digraph_cosine(digraph_anon,
    digraph_jop))
    print("similarity btw Jop and Mom: ", compare_digraph_cosine(digraph_mom,
    digraph_jop))

    print("similarity to Jop: ", diff_jop)
    print("similarity to Anon: ", diff_anon)
    print("similarity to Mom: ", diff_mom)
```

In our program, we have the user type the same message out 5 times and then average the time. Then, we compare the user's digraph to our existing digraphs. Here is some example output.

Output at the digraph recording step:



```
Input your name (this is used for identification). jop3

Type this paragraph, then press esc. on finish: (do not mind errors)

the quick brown fox jumps over the lazy dog.
th 174.62074756622314 2
he 192.0710802078247 2
espace 191.81323051452637 2
spaceq 256.192684173584 1
qu 177.1554946899414 1
ui 174.35503005981445 1
ic 111.50026321411133 1
ck 436.1569881439209 1
kspace 180.27639389038086 1
spaceb 234.83586311340332 1
br 209.84983444213867 1
ro 126.24764442443848 1
ow 241.99938774108887 1
wn 133.3012580871582 1
nspace 181.33163452148438 1
spacef 189.70632553100586 1
fo 215.99173545837402 1
ox 141.1302089691162 1
xspace 147.6759910583496 1
spacej 224.55716133117676 1
ju 137.89701461791992 1
um 198.1065273284912 1
mp 228.13987731933594 1
ps 97.81098365783691 1
sspace 211.68804168701172 1
spaceo 108.02245140075684 1
ov 309.76176261901855 1
ve 136.69776916503906 1
```

Output at the comparison step: 

```
spaceb 35.21941850582759 4
br 136.20416820049286 4
ro 60.25421619415283 4
ow 78.01271875699362 5
wn 35.88471213976542 5
nspace 54.931809504826866 5
spacef 97.81342082553441 6
fo 39.92724948459201 6
ox 45.311244328816734 6
xspace 63.06236812046596 7
spacej 184.55040256182355 5
ju 57.83037940661112 5
um 56.09501600265503 5
mp 60.140420993169144 5
ps 10.451088349024456 4
sspace 22.11753030618032 4
spaceo 57.764941453933716 5
ov 39.436084032058716 5
ve 50.97723404566447 5
er 22.27567632993O626 5
rspace 51.147156953811646 5
spacet 45.02941767374675 5
spacel 75.50129294395447 5
la 33.496512969334916 5
az 56.067005793253585 5
zy 41.32969776789347 5
yspace 48.72429370880127 5
spaced 94.48591868082683 5
do 46.06159329414368 5
og 54.28927540779114 5
g. 191.2211736043294 5
.esc 408.3331147829691 5
_total 3943.59180132548 5
bo 447.98779487609863 1
uc 467.68295764923096 2
ci 224.90406036376953 1
ik 281.3107967376709 1
kb 1704.1542530059814 1
px 93.12939643859863 1
similarity btw Jop and Anon:  0.835250138032722
similarity btw Jop and Mom:  0.9813458103594435
similarity to Jop:  0.9914681792119782
similarity to Anon:  0.6675736609300276
similarity to Mom:  0.9803060343692975
```

The user here is Jop. We can see that this program can recognize Jop from Anon, since there is considerable difference between Jop and Anon's digraph. However the program can hardly separate Jop from Mom because their digraphs are too similar. Thus our program still has limited capability.

1. How many words do we need to correctly identify the person?

Normally, in a real world use case like https://www.typingdna.com/verify, 2-factor authentication uses around 4 words to identify the keystroke pattern.

2. Do you think this method is scalable? (to thousand persons) for either recognition system or identification system. Please provide your analysis.

According to this paper, keystrokes dynamics has been reported to be around 5.4% in terms of equal error rate (EER), as compared to 0.2% for fingerprints, which does not scale very well. If normal fingerprints should be used within the range of 1000-5000 people, the keystrokes shoyld be used around a hundred people. Keystrokes should be used as an additional layers of authentication.

3. Will you use this kind of authentication in your system? Please also provide a reason.

As mentioned in the paper above, the keystrokes have much higher error rate than a fingerprints. It may works if our system is used within a hundred users and username and password are not desired. Keystrokes are also cheap because the only need is a keyboard. But as mentioned, keystrokes should be used only as an additional layers of authentication.