AI_ML_II-dstrube3

October 25, 2022

[10]:
```python
# If running for the first time:
# %pip install gensim
# %pip install opencv-python

from gensim.models import Word2Vec
import gensim.models
import pandas as pd
import numpy as np
import cv2
import os
import time
import re

newmodel = gensim.models.KeyedVectors.load_word2vec_format('reducedvector.bin',␣
↪binary=True)
"""
# Find the five nearest neighbors to the word man
print('Five nearest neighbors to the word man: ')
print(newmodel.most_similar('man', topn=5))
# [('woman', 0.5876938104629517), ('girl', 0.5229198932647705), ('young', 0.
↪49715912342071533), ('immortal', 0.4890766441822052), ('spider', 0.
↪47930291295051575)]

# Compute a measure of similarity between woman and man
print('A measure of similarity between woman and man: ')
print(newmodel.similarity('woman', 'man'))
# 0.5876938

# To complete analogies like man is to woman as king is to ??, we can use:
print('Man is to woman as king is to ??:')
print(newmodel.most_similar(positive=['king', 'woman'], negative=['man'],␣
↪topn=1))
# [('queen', 0.5532454252243042)]

"""
print('Setup complete')
```

```
Setup complete
```

Task Set #1

Q1: We will use the target words - man and woman. Use the pre-trained word2vec model to rank the following 15 words from the most similar to the least similar to each target word. For each word-target word pair, provide the similarity score. Provide your results in table format.

[2]:
```
input_list=['wife', 'husband', 'child', 'queen', 'king', 'man', 'woman',␣
↪'birth', 'doctor', 'nurse', 'teacher',
            'professor', 'engineer', 'scientist', 'president']
man_dict = {}
woman_dict = {}
for item in input_list:
    man_value = newmodel.similarity('man', item)
    woman_value = newmodel.similarity('woman', item)
    man_dict[item] = man_value
    woman_dict[item] = woman_value


def print_table_from_dict(dictionary, name):
    # https://www.tutorialsteacher.com/articles/sort-dict-by-value-in-python
    ionary = sorted(dictionary.items(), key=lambda x:x[1], reverse=True)
    # ^^^ was going to call this d_list, but then I thought of this funny thing:
    dictionary = dict(ionary)

    # doesn't really look like a table using either of these:
    # print(dictionary)
    # display(dictionary)

    # Must re-index to put into a dataframe:
    dictionary = {'word':dictionary.keys(), 'score':dictionary.values()}
    # https://www.geeksforgeeks.org/
↪how-to-convert-dictionary-to-pandas-dataframe/
    data = pd.DataFrame.from_dict(dictionary)
    print(f"Sorted {name} list:")
    display(data)
    print()

print_table_from_dict(man_dict, 'man')
print_table_from_dict(woman_dict, 'woman')
```

Sorted man list:

	word	score
0	man	1.000000
1	woman	0.587694
2	child	0.333422
3	doctor	0.289247
4	wife	0.283479
5	king	0.264497

```
6     husband  0.234116
7       nurse  0.153481
8       birth  0.123439
9   scientist  0.112269
10      queen  0.110419
11  professor  0.107622
12    teacher  0.098740
13  president  0.094579
14   engineer  0.087364


Sorted woman list:
         word     score
0       woman  1.000000
1       child  0.589809
2         man  0.587694
3     husband  0.449643
4       birth  0.420309
5        wife  0.300689
6       nurse  0.254358
7       queen  0.228572
8     teacher  0.204078
9      doctor  0.196134
10  scientist  0.137311
11       king  0.122529
12  professor  0.105199
13  president  0.084627
14   engineer  0.044264
```

Q2: The Bigger Analogy Test Set (BATS) Word analogy task has been one of the standard benchmarks for word embeddings since 2013 (https://vecto.space/projects/BATS/).

A) Select any file from the downloaded dataset (BATS_3.0.zip). For each row in your selected file, choose a target word from the row and provide the measure of similarity between your target word and the other words on the row (Remember to document the file used).

[3]:
```
file_path = 'BATS_3.0/3_Encyclopedic_semantics/E09 [things - color].txt'
data = pd.read_csv(file_path, sep="\s+", header=None).rename({0: "target", 1:
 ↪"words"}, axis=1)

data_list = [(data.at[index, "target"], data.at[index, "words"].split("/")) for␣
 ↪index in range(data.shape[0])]

scores = {}
index = 0
for target, words in data_list:
    for word in words:
```

```
        try:
            score = (target, word, newmodel.similarity(target, word))
            scores[index] = score
            index += 1
        except KeyError:
            # print(f"{word} not found.")
            continue

print(f"Selected file: {file_path}:\n")

data = pd.DataFrame.from_dict(scores)
# Transpose
data = data.T

# display(data)  # <- good for a truncated print out
# https://www.geeksforgeeks.org/
 ↪how-to-print-an-entire-pandas-dataframe-in-python/
# display(data.to_string())  # <- looks weird
print(data.to_string())
```

```
Selected file: BATS_3.0/3_Encyclopedic_semantics/E09 [things - color].txt:

              0          1         2
0           ant      black  0.016854
1           ant      brown  0.164345
2           ant        red  0.086045
3         apple        red  0.169385
4         apple     orange  0.204806
5         apple     yellow  0.117173
6         apple     golden  0.149683
7    blackboard      black  0.002111
8    blackboard      green  0.038799
9         blood        red  0.239439
10    blueberry       blue  0.145673
11    blueberry      black  0.102351
12     broccoli      green  0.220847
13       bruise       blue  0.250219
14       bruise     purple  0.402197
15      cabbage      green   0.21119
16       carrot     orange  0.199563
17       carrot        red  0.135183
18       carrot     yellow  0.237842
19  cauliflower      white  0.178381
20  cauliflower      green  0.287544
21  cauliflower     yellow  0.309281
22  cauliflower  yellowish  0.464636
23       celery      green  0.272918
```

24	celery	white	0.198108
25	celery	brown	0.281928
26	cherry	red	0.260896
27	cherry	yellow	0.313398
28	cherry	black	0.221335
29	chocolate	white	0.30966
30	chocolate	brown	0.313166
31	chocolate	black	0.276576
32	cloud	white	0.206877
33	cloud	gray	0.208631
34	cloud	grey	0.144232
35	coal	black	0.072756
36	coffee	black	0.186443
37	coffee	brown	0.204282
38	cranberry	red	0.125799
39	cranberry	purple	0.273153
40	cranberry	pink	0.297515
41	cream	white	0.334988
42	crow	black	0.231378
43	cucumber	green	0.231164
44	emerald	green	0.430846
45	frog	green	0.368224
46	frog	brown	0.401646
47	frog	grey	0.373885
48	frog	gray	0.420906
49	grapes	black	0.22214
50	grapes	red	0.193513
51	grapes	green	0.234692
52	grapes	purple	0.363582
53	grass	green	0.393804
54	leaves	green	0.38611
55	leaves	red	0.239475
56	leaves	yellow	0.3729
57	milk	white	0.226181
58	paper	white	0.243027
59	paper	color	0.299122
60	parsley	green	0.221835
61	pepper	black	0.305234
62	pepper	red	0.156652
63	pepper	green	0.26711
64	pepper	yellow	0.261175
65	pepper	orange	0.30587
66	potato	brown	0.205794
67	radish	red	0.188842
68	radish	pink	0.379294
69	radish	white	0.23589
70	radish	green	0.260944
71	radish	black	0.217528

```
72          raven      black  0.152673
73           rose        red  0.129654
74           rose     yellow  0.123819
75           rose       pink  0.183255
76           rose      white  0.201757
77           rose       blue  0.162757
78           ruby        red  0.126898
79           salt      white  0.178166
80       sapphire       blue  0.357904
81            sea       blue  0.195446
82            sea      green  0.182578
83            sea       gray  0.065837
84            sea       grey  0.124886
85            sky       blue   0.44397
86            sky       gray  0.167499
87            sky       grey  0.217461
88           snow      white  0.385684
89           soil      black  0.169323
90           soil      brown  0.127698
91           soil       dark  0.137971
92        spinach      green  0.170907
93          sugar      white  0.190748
94          sugar      brown  0.170382
95            sun     yellow  0.152244
96            sun       gold  0.029353
97           swan      white  0.395133
98           swan      black   0.33638
99           swan       gray  0.461392
100          swan       grey  0.425437
101           tea      black  0.245288
102           tea      green  0.402273
103           tea      white  0.283473
104           tea        red  0.316095
105           tea      brown   0.22695
106           tea     yellow  0.344161
107        tomato        red  0.139281
108    toothpaste      white  0.096497
109       yoghurt      white  0.092022
110       yoghurt       pink  0.254411
```

Q2 B) Think of three words that identify membership in one of the protected classes (choose only one class): race, color, religion, or national origin. For each row in your selected BATS_3.0 file, compute the similarity between your target word and each of your three words. Indicate when there are noticeable differences in the similarity scores based on membership in the protected class. Provide your results in table format.

[4]:
```python
#Protected class: national origin
protected_class_words = ['african', 'american', 'chinese']
```

```
scores = {}
index = 0
for target, _ in data_list:
    for word in protected_class_words:
        try:
            score = (target, word, newmodel.similarity(target, word))
            scores[index] = score
            index += 1
        except KeyError:
            # print(f"{word} not found.")
            continue

data = pd.DataFrame.from_dict(scores)

data = data.T

print('Similarity between target word and each of three words from protected␣
 ↪class: national origin:')
print(data.to_string())

index = 0
tuple_list = []
noticeable_differences = []
for key in scores.keys():
    item = scores[key]
    tuple_list.append(item)
    index += 1
    if index % 3 == 0:
        value_0 = tuple_list[0][2]
        value_1 = tuple_list[1][2]
        value_2 = tuple_list[2][2]
        # Significant: one is negative and the others are positive, or vice␣
 ↪versa
        # There are other forms of significance, but they are subtler and␣
 ↪harder to distinguish programmatically;
        # this should be good enough for now
        if value_0 < 0 and value_1 > 0 and value_2 > 0:  # Only 0 -
            noticeable_differences.append(tuple_list[0])
        elif value_0 > 0 and value_1 < 0 and value_2 > 0:  # Only 1 -
            noticeable_differences.append(tuple_list[1])
        elif value_0 > 0 and value_1 > 0 and value_2 < 0:  # Only 2 -
            noticeable_differences.append(tuple_list[2])
        elif value_0 < 0 and value_1 < 0 and value_2 > 0:  # Only 2 +
            noticeable_differences.append(tuple_list[2])
        elif value_0 < 0 and value_1 > 0 and value_2 < 0:  # Only 1 +
            noticeable_differences.append(tuple_list[1])
```

```python
        elif value_0 > 0 and value_1 < 0 and value_2 < 0:  # Only 0 +
            noticeable_differences.append(tuple_list[0])
        tuple_list = []
df_nd = pd.DataFrame(noticeable_differences)
print('\n\nNoticeable differences: ', end='')
display(df_nd)
```

```
Similarity between target word and each of three words from protected class:
national origin:
              0         1         2
0           ant   african  -0.02331
1           ant  american -0.014413
2           ant   chinese -0.043652
3         apple   african -0.108922
4         apple  american -0.031242
5         apple   chinese -0.001866
6    blackboard   african -0.118044
7    blackboard  american -0.069308
8    blackboard   chinese -0.025206
9         blood   african  0.020525
10        blood  american -0.065638
11        blood   chinese -0.001763
12    blueberry   african  0.107284
13    blueberry  american -0.084573
14    blueberry   chinese  0.023756
15     broccoli   african -0.019525
16     broccoli  american -0.006587
17     broccoli   chinese -0.084253
18       bruise   african -0.018624
19       bruise  american -0.051657
20       bruise   chinese -0.142612
21      cabbage   african  0.027477
22      cabbage  american -0.066327
23      cabbage   chinese  0.029704
24       carrot   african  -0.05758
25       carrot  american -0.091398
26       carrot   chinese -0.043109
27  cauliflower   african  0.050677
28  cauliflower  american  0.019855
29  cauliflower   chinese  0.042827
30       celery   african  0.045113
31       celery  american -0.059775
32       celery   chinese  0.100531
33       cherry   african  0.010406
34       cherry  american  0.016308
35       cherry   chinese -0.041911
36    chocolate   african -0.043909
```

37	chocolate	american	-0.011377
38	chocolate	chinese	0.088184
39	cloud	african	-0.120791
40	cloud	american	-0.086336
41	cloud	chinese	-0.151692
42	coal	african	0.043159
43	coal	american	-0.063821
44	coal	chinese	-0.102578
45	coffee	african	0.095358
46	coffee	american	0.020197
47	coffee	chinese	0.040352
48	cranberry	african	0.078812
49	cranberry	american	-0.058909
50	cranberry	chinese	0.065395
51	cream	african	0.03126
52	cream	american	-0.013076
53	cream	chinese	0.06882
54	crow	african	0.129586
55	crow	american	0.192022
56	crow	chinese	0.049833
57	cucumber	african	-0.022711
58	cucumber	american	-0.062825
59	cucumber	chinese	-0.041572
60	emerald	african	0.059782
61	emerald	american	0.024662
62	emerald	chinese	-0.026351
63	frog	african	-0.015949
64	frog	american	-0.001689
65	frog	chinese	-0.024228
66	grapes	african	-0.043244
67	grapes	american	-0.106721
68	grapes	chinese	-0.042963
69	grass	african	0.021803
70	grass	american	-0.078633
71	grass	chinese	0.004732
72	leaves	african	-0.015702
73	leaves	american	-0.10596
74	leaves	chinese	-0.015529
75	milk	african	-0.00043
76	milk	american	-0.023517
77	milk	chinese	0.092944
78	paper	african	-0.002318
79	paper	american	-0.01711
80	paper	chinese	0.004734
81	parsley	african	-0.042133
82	parsley	american	-0.058983
83	parsley	chinese	0.020827
84	pepper	african	0.11233

85	pepper	american	0.010069
86	pepper	chinese	0.030423
87	potato	african	0.065262
88	potato	american	0.023001
89	potato	chinese	0.045317
90	radish	african	0.022052
91	radish	american	-0.074765
92	radish	chinese	0.070388
93	raven	african	-0.003932
94	raven	american	0.066115
95	raven	chinese	0.00835
96	rose	african	0.182046
97	rose	american	0.151537
98	rose	chinese	-0.037605
99	ruby	african	-0.001984
100	ruby	american	0.109451
101	ruby	chinese	-0.079896
102	salt	african	0.008389
103	salt	american	0.005882
104	salt	chinese	-0.028198
105	sapphire	african	-0.030863
106	sapphire	american	0.041551
107	sapphire	chinese	-0.023181
108	sea	african	0.104766
109	sea	american	-0.050821
110	sea	chinese	-0.01741
111	sky	african	-0.029269
112	sky	american	-0.018682
113	sky	chinese	-0.032061
114	snow	african	0.023703
115	snow	american	-0.005688
116	snow	chinese	-0.019209
117	soil	african	0.044015
118	soil	american	0.037046
119	soil	chinese	-0.072177
120	spinach	african	0.003485
121	spinach	american	-0.068013
122	spinach	chinese	0.041408
123	sugar	african	0.078022
124	sugar	american	0.009119
125	sugar	chinese	0.040515
126	sun	african	-0.045166
127	sun	american	-0.020564
128	sun	chinese	0.115022
129	swan	african	-0.002829
130	swan	american	0.07223
131	swan	chinese	-0.026455
132	tea	african	0.141133

133	tea	american	0.094102
134	tea	chinese	0.163353
135	tomato	african	0.013456
136	tomato	american	-0.0725
137	tomato	chinese	0.054584
138	toothpaste	african	0.028104
139	toothpaste	american	-0.037669
140	toothpaste	chinese	0.01675
141	yoghurt	african	0.043475
142	yoghurt	american	-0.125141
143	yoghurt	chinese	-0.028097

Noticeable differences:

	0	1	2
0	blood	african	0.020525
1	blueberry	american	-0.084573
2	cabbage	american	-0.066327
3	celery	american	-0.059775
4	cherry	chinese	-0.041911
5	chocolate	chinese	0.088184
6	coal	african	0.043159
7	cranberry	american	-0.058909
8	cream	american	-0.013076
9	emerald	chinese	-0.026351
10	grass	american	-0.078633
11	milk	chinese	0.092944
12	paper	chinese	0.004734
13	parsley	chinese	0.020827
14	radish	american	-0.074765
15	raven	african	-0.003932
16	rose	chinese	-0.037605
17	ruby	american	0.109451
18	salt	chinese	-0.028198
19	sapphire	american	0.041551
20	sea	african	0.104766
21	snow	african	0.023703
22	soil	chinese	-0.072177
23	spinach	american	-0.068013
24	sun	chinese	0.115022
25	swan	american	0.072230
26	tomato	american	-0.072500
27	toothpaste	american	-0.037669
28	yoghurt	african	0.043475

Q3: Sentences:

a. Complete some sentences with your own word analogies. Use the Word2Vec model to find the similarity measure between your pair of words. Provide your results.

[5]:
```
"""
1- king is to throne as judge is to ___? bench
2- giant is to dwarf as genius is to ___? moron
3- college is to dean as jail is to ___? warden
4- arc is to circle as line is to ___? polygon
5- French is to France as Dutch is to ___? Netherlands
6- man is to woman as king is to ___? queen
7- water is to ice as liquid is to ___? solid
8- bad is to good as sad is to ___? happy
9- nurse is to hospital as teacher is to ___? school
10- usa is to pizza as japan is to ___? sushi
11- human is to house as dog is to ___? shed
12- grass is to green as sky is to ___? blue
13- video is to cassette as computer is to ____? floppy
14- universe is to planet as house is to ____? atom
15- poverty is to wealth as sickness is to ___? health
"""

all_pairs = []
all_pairs.append(['king','throne','judge','bench'])
all_pairs.append(['giant','dwarf','genius','moron'])
all_pairs.append(['college','dean','jail','warden'])
all_pairs.append(['arc','circle','line','polygon'])
all_pairs.append(['french','france','dutch','netherlands'])
all_pairs.append(['man','woman','king','queen'])
all_pairs.append(['water','ice','liquid','solid'])
all_pairs.append(['bad','good','sad','happy'])
all_pairs.append(['nurse','hospital','teacher','school'])
all_pairs.append(['usa','pizza','japan','sushi'])
all_pairs.append(['human','house','dog','shed'])
all_pairs.append(['grass','green','sky','blue'])
all_pairs.append(['video','cassette','computer','floppy'])
all_pairs.append(['universe','planet','house','atom'])
all_pairs.append(['poverty','wealth','sickness','health'])
my_scores = []
for tup in all_pairs:
    score = newmodel.similarity(tup[2], tup[3])
    print(tup[2] + ', ' + tup[3] + ' -> ' + str(score))
    my_scores.append(score)
```

```
judge, bench -> 0.30267337
genius, moron -> 0.12846608
jail, warden -> 0.27777424
line, polygon -> 0.22809683
dutch, netherlands -> 0.41922888
king, queen -> 0.5685571
```

```
liquid, solid -> 0.6546474
sad, happy -> 0.44885093
teacher, school -> 0.5326567
japan, sushi -> 0.01186634
dog, shed -> 0.10359062
sky, blue -> 0.4439698
computer, floppy -> 0.32768583
house, atom -> 0.07146792
sickness, health -> 0.19527604
```

Q3 b. Use the Word2Vec model to find the word analogy and corresponding similarity score. Provide your results.

[6]:
```
Word2Vec_scores = []
for tup in all_pairs:
    print(f"{tup[0]} is to {tup[1]} as {tup[2]} is to : ", end='')
    result = newmodel.most_similar(positive=[tup[2], tup[1]],␣
 ↪negative=[tup[0]], topn=1)
    print(result)
    score = result[0][1]
    Word2Vec_scores.append(score)
```

```
king is to throne as judge is to : [('prosecution', 0.5186458230018616)]
giant is to dwarf as genius is to : [('theorist', 0.4280889630317688)]
college is to dean as jail is to : [('peress', 0.5444425940513611)]
arc is to circle as line is to : [('lines', 0.4287526607513428)]
french is to france as dutch is to : [('netherlands', 0.6044681072235107)]
man is to woman as king is to : [('queen', 0.5532454252243042)]
water is to ice as liquid is to : [('solid', 0.4500039219856262)]
bad is to good as sad is to : [('glory', 0.440381795167923)]
nurse is to hospital as teacher is to : [('institution', 0.48289817571640015)]
usa is to pizza as japan is to : [('dishes', 0.5763506293296814)]
human is to house as dog is to : [('hound', 0.4231664538383484)]
grass is to green as sky is to : [('blue', 0.5478643178939819)]
video is to cassette as computer is to : [('peripherals', 0.6654507517814636)]
universe is to planet as house is to : [('houses', 0.4264702796936035)]
poverty is to wealth as sickness is to : [('impious', 0.49606096744537354)]
```

Q3 c Lastly, compute and print the correlation between the vector of similarity scores from your analogies versus the Word2Vec analogy-generated similarity scores. What is the strength of the correlation?

- .00-.19 “very weak” correlation
- .20-.39 “weak” correlation
- .40-.59 “moderate” correlation
- .60-.79 “strong” correlation
- .80-1.0 “very strong” correlation

[7]:
```python
for index in range(len(all_pairs)):
    print(f"my score: {my_scores[index]:3f} -> Word2Vec's score:␣
↪{Word2Vec_scores[index]:3f}")
    index += 1

# https://numpy.org/doc/stable/reference/generated/numpy.corrcoef.html
print('\nCorrelation coefficient: ', end='')
print(np.corrcoef(my_scores, Word2Vec_scores)[0][1])

# Copied from my A3 - AI ML 1:
def correlation_category(correlation_value):
    output = ''
    if abs(correlation_value) < 0.2:
        output += 'Very weak correlation'
    elif abs(correlation_value) < 0.4:
        output += 'Weak correlation'
    elif abs(correlation_value) < 0.6:
        output += 'Moderate correlation'
    elif abs(correlation_value) < 0.8:
        output += 'Strong correlation'
    else:
        output += 'Very strong correlation'
    return output

print('\nCorrelation category: ' + correlation_category(np.corrcoef(my_scores,␣
↪Word2Vec_scores)[0][1]))
```

```
my score: 0.302673 -> Word2Vec's score: 0.518646
my score: 0.128466 -> Word2Vec's score: 0.428089
my score: 0.277774 -> Word2Vec's score: 0.544443
my score: 0.228097 -> Word2Vec's score: 0.428753
my score: 0.419229 -> Word2Vec's score: 0.604468
my score: 0.568557 -> Word2Vec's score: 0.553245
my score: 0.654647 -> Word2Vec's score: 0.450004
my score: 0.448851 -> Word2Vec's score: 0.440382
my score: 0.532657 -> Word2Vec's score: 0.482898
my score: 0.011866 -> Word2Vec's score: 0.576351
my score: 0.103591 -> Word2Vec's score: 0.423166
my score: 0.443970 -> Word2Vec's score: 0.547864
my score: 0.327686 -> Word2Vec's score: 0.665451
my score: 0.071468 -> Word2Vec's score: 0.426470
my score: 0.195276 -> Word2Vec's score: 0.496061

Correlation coefficient: 0.16294155504979496

Correlation category: Very weak correlation
```

Task Set #2

Q1: Each image in the dataset has a unique value representing age, gender, and race based on the following legend:

- age: indicates the age of the person in the picture and can range from 0 to 116.
- gender: indicates the gender of the person and is either 0 (male) or 1 (female).
- race: indicates the race of the person and can from 0 to 4, denoting White, Black, Asian, Indian, and Others (like Hispanic, Latino, Middle Eastern).

[8]:
```
# https://pyimagesearch.com/2014/09/15/python-compare-two-images/
def mse(vals_1, vals_2):
    # the 'Mean Squared Error' between the two images is the
    # sum of the squared difference between the two images;
    # NOTE: the two images must have the same dimension
    err = np.sum((vals_1[0] - vals_2[0]) ** 2)
    err /= float(vals_1[1] * vals_1[2])

    # return the MSE, the lower the error, the more "similar"
    # the two images are
    return err


def get_images_values(files, folder_path):
    image_values = {}

    for image_file in files:
        image = cv2.imread(folder_path + image_file)
        if image is not None:
            image_values[image_file] = [image.astype('float'), image.shape[0],␣
↪image.shape[1]]
    return image_values


def get_time(seconds):
    # Copied from my CS 7641 Assignment 2
    if int(seconds / 60) == 0:
        if int(seconds) == 0:
            return str(round(seconds, 3)) + ' seconds'
        return str(int(seconds)) + ' second(s)'
    minutes = int(seconds / 60)
    seconds = int(seconds % 60)
    if int(minutes / 60) == 0:
        return str(minutes) + ' minute(s) and ' + str(seconds) + ' second(s)'
    hours = int(minutes / 60)
    minutes = int(minutes % 60)
    # Assuming this won't be called for any time span greater than 24 hours
    return str(hours) + ' hour(s), ' + str(minutes) + ' minute(s), and ' +␣
↪str(seconds) + ' second(s)'
```

```python
path = 'crop_part1/'
dir_list = os.listdir(path)
dir_list.sort()
compare_list = os.listdir(path)
compare_list.sort()
images_values = get_images_values(dir_list, path)

print('files count: ' + str(len(dir_list)))
"""
# fileA = '96_1_0_20170110183855839.jpg.chip.jpg'  # same
# fileB = '96_1_1_20170110183853718.jpg.chip.jpg'  # same
# fileC = '96_1_0_20170110182515404.jpg.chip.jpg'  # different
# m = mse(images_values[fileA], images_values[fileB])
# print('mse (A & B): ' + str(m))  # 0.0
# m = mse(images_values[fileA], images_values[fileC])
# print('mse (A & C): ' + str(m))  # 7251.330525

# fileA = '90_1_0_20170110182841384.jpg.chip.jpg'  # same, but blurry :/, mse >␣
↪100
# fileB = '96_1_0_20170110182019881.jpg.chip.jpg'
# m = mse(fileA, fileB, path)
# print('mse (A & B): ' + str(m)) # MSE = 187.775225
# Taking this as a baseline for nearly identical images
"""

duplicates = []
anomalies = []
index = 0
print('Looking for duplicates & bad file names. "." = 1 file checked. "#" =␣
↪duplicate found...')
start = time.time()
for file in dir_list:
    if file not in images_values.keys():
        # File didn't make it into images_values (like .DS_Store) - skip it
        compare_list.remove(file)
        anomalies.append(file)
        continue
    file_parts = file.split('_')
    if len(file_parts) < 4:
        print('\nbad file name: ' + file)
    if file not in compare_list:
        # Previously detected duplicate
        continue
    compare_list.remove(file)
    for duplicate in duplicates:
        if duplicate in compare_list:
```

```
            compare_list.remove(duplicate)
    for file_other in compare_list:
        m = mse(images_values[file], images_values[file_other])
        if m < 188:
            # print(file + ' seems to be a duplicate of ' + file_other)
            print('#', end='')
            # duplicates.append(file)
            duplicates.append(file_other)
            """
            12 duplicates found below 0.35% (unsorted), including:
            21_0_4_20161223214826657.jpg.chip.jpg, 6_1_4_20170103230723185.jpg.chip.jpg
            I highly doubt subject is really 21 years old in this picture
            """
    print('.', end='')
    index += 1
    if index % 100 == 0:
        print()

end = time.time()
print('\nDone in ' + get_time(end - start))
print('Found ' + str(len(duplicates)) + ' duplicates.')
for duplicate in duplicates:
    if duplicate in dir_list:
        dir_list.remove(duplicate)

for anomaly in anomalies:
    if anomaly in dir_list:
        dir_list.remove(anomaly)

# Data has been cleaned up some. (Could probably do more, but this should be good enough for now)
```

```
files count: 9781
Looking for duplicates & bad file names. "." = 1 file checked. "#" = duplicate found…
…#…#…#..#..#.#.#..#…
…#…#…
…#…#…##…#…#…#..#…
…#…
.#…#…#…#
…#..#…
…
#…
…#…#…
…
…#…#…
```

```
…#…#.#…#…#..
…#…#…
…#.
…#…#…#…
…
…#…#…
…
…#…
…
…#…#…
…
…#…#…
…#…
…#…##…#…##…
…#…
…#…#…#…
…
…#…#…#…
…
…#…#
…
…#…##…#..#…
…##…#…
..#…##…#…
…
.#…#…#.#…#…
…#…
…#…##…#…#…
…
…#…
…
…#…#…#…
…
…#…#.#…
…
…###…#…#…#.#…
…#.
.#.#.#..#.#.#.##.#…#…#…
…
…#…
…
…#…#…#…
…
…#…
…#…#…
…
…
..#…
```

…#…
…
…
…
…
…
…
…#…
…
…
…
…#…
…
…
…
…#…
…#…
…
…
…#…#…#…
…
…#…
…
…#…#…
…#…
…#…#…
…
…#…
…
…#…#…
…
…
…
…
…
…#…
…
…
…
…#…
…
…#…
…
…#…
…
…
.#…
…

```
...
...#...#...
...#.#...#..
...
...
...
...
...
...
...
...
...#...
...
...
...
...#...
...#...
...
...#...
...#...
...
...#..#...
...
...
...
...###...#...
...
...
...##..
...#...#.#...
...
...#...
...
...#...#...
...
...#...
...
...#...
...
...#...#.
...#...
...
...#...
...#...#...
...#...
...#..#...#...#...
...
...
```

```
bad file name: 61_1_20170109142408075.jpg.chip.jpg
..#..#..
bad file name: 61_3_20170109150557335.jpg.chip.jpg
…##.#…#…
..#…
…
…#…#..
…
…#…#…
#…
…#..##…
…
…#…#…
…
#…#…##…
…#…
…#…#…#…
…
…#…#…#…#.
…
…
…
…#…#…
..#…
..#…#…#…
…
…##…
…
…
…
…#…#…
.#…#…
…
…
…#…
…
…#…#…
…
…
Done in 3 hour(s), 17 minute(s), and 31 second(s)
Found 224 duplicates.
```

Compute and document the frequency of images associated with each subgroup for age (subdivide based on - (0-20), (21,40), (41,60), (61,80), (81, 116)), gender (0,1), and race (0 to 4). Which subgroup in each age, gender, and race category has the largest representation? Which subgroup in each age, gender, and race category has the least representation?

[12]:

```
# Sum up counts for each group...
# age_0_20 = 0
# age_21_40 = 0
# age_41_60 = 0
# age_61_80 = 0
# age_81_116 = 0
# male = 0
# female = 0
# white = 0
# black = 0
# asian = 0
# indian = 0
# others = 0

# for file in dir_list:
#     file_parts = file.split('_')
#     age = int(file_parts[0])
#     if age < 21:
#         age_0_20 += 1
#     elif age < 41:
#         age_21_40 += 1
#     elif age < 61:
#         age_41_60 += 1
#     elif age < 81:
#         age_61_80 += 1
#     else:
#         age_81_116 += 1
#
#     if len(file_parts) < 4:
#         continue
#
#     gender = int(file_parts[1])
#     if gender == 0:
#         male += 1
#     else:
#         female += 1
#
#     race = int(file_parts[2])
#     if race == 0:
#         white += 1
#     elif race == 1:
#         black += 1
#     elif race == 2:
#         asian += 1
#     elif race == 3:
#         indian += 1
#     else:
```

```python
#         others += 1

# On second thought, this is getting too tedious. Let's try to do this a little more smartly...
genders = {0: 'male', 1: 'female'}
races = {0: 'white', 1: 'black', 2: 'asian', 3: 'indian', 4: 'other'}
img_names = []
for file in dir_list:
    img_names.append(file.replace('.jpg.chip.jpg', ''))

age_pattern = re.compile(r"(^\d{1,3}).+")
gender_pattern = re.compile(r"^\d{1,3}_(\d).+")
race_pattern = re.compile(r"^\d{1,3}_\d_(\d).+")
age_list = []
gender_list = []
race_list = []
for name in img_names:
    age_list.append(int(re.match(age_pattern, name).group(1)))
    gender_list.append(int(re.match(gender_pattern, name).group(1)))
    race_list.append(int(re.match(race_pattern, name).group(1)))

img_df = pd.DataFrame({
    "img_name": img_names, "ages": age_list,
    "genders": gender_list, "races": race_list
    })

bins = pd.IntervalIndex.from_tuples([(0, 20), (21, 40), (41, 60), (61, 80), (81, 116)])
img_df["age"] = pd.cut(img_df["ages"], bins)

img_df["gender"] = img_df["genders"].map(genders)

img_df["race"] = img_df["races"].map(races)

df = img_df[["age", "gender", "race"]]

age_value_counts = df["age"].value_counts()
age_group_max = age_value_counts[age_value_counts == age_value_counts.max()]
age_proportion_max = 100 * (age_group_max.values[0] / age_value_counts.sum())
age_group_min = age_value_counts[age_value_counts == age_value_counts.min()]
age_proportion_min = 100 * (age_group_min.values[0] / age_value_counts.sum())
print(f"Age group with largest representation: {age_group_max.index[0]} ({age_proportion_max:.2f}%)")
print(f"Age group with least representation: {age_group_min.index[0]} ({age_proportion_min:.2f}%)")

gender_value_counts = df["gender"].value_counts()
```

```
gender_group_max = gender_value_counts[gender_value_counts ==␣
↪gender_value_counts.max()]
gender_proportion_max = 100 * (gender_group_max.values[0] / gender_value_counts.
↪sum())
gender_group_min = gender_value_counts[gender_value_counts ==␣
↪gender_value_counts.min()]
gender_proportion_min = 100 * (gender_group_min.values[0] / gender_value_counts.
↪sum())
print(f"Gender group with largest representation: {gender_group_max.index[0]}␣
↪({gender_proportion_max:.2f}%)")
print(f"Gender group with least representation: {gender_group_min.index[0]}␣
↪({gender_proportion_min:.2f}%)")

race_value_counts = df["race"].value_counts()
race_group_max = race_value_counts[race_value_counts == race_value_counts.max()]
race_proportion_max = 100 * (race_group_max.values[0] / race_value_counts.sum())
race_group_min = race_value_counts[race_value_counts == race_value_counts.min()]
race_proportion_min = 100 * (race_group_min.values[0] / race_value_counts.sum())
print(f"Race group with largest representation: {race_group_max.index[0]}␣
↪({race_proportion_max:.2f}%)")
print(f"Race group with least representation: {race_group_min.index[0]}␣
↪({race_proportion_min:.2f}%)")
```

```
Age group with largest representation: (0, 20] (44.45%)
Age group with least representation: (81, 116] (3.54%)
Gender group with largest representation: female (55.07%)
Gender group with least representation: male (44.93%)
Race group with largest representation: white (53.83%)
Race group with least representation: black (4.16%)
```

Recreate a table of the age group, gender, and race distributions of subjects based on the UTK dataset subgroups (inspired by the one discussed in the lecture and reposted below). Based on what you've learned so far, if an algorithm is trained based on this dataset, which group(s) will be impacted the most? Explain why.

[18]:

```
age_gender_table = df[["age", "gender"]].value_counts().reset_index(drop=False).
↪rename({0: "n"}, axis=1)
age_gender_table = age_gender_table.pivot(index="gender", columns="age",␣
↪values="n").rename_axis(None)
age_gender_table.columns.name = None

age_race_table = df[["age", "race"]].value_counts().reset_index(drop=False).
↪rename({0: "n"}, axis=1)
age_race_table = age_race_table.pivot(index="race", columns="age", values="n").
↪rename_axis(None)
age_race_table.columns.name = None
```

```python
table = pd.concat([age_gender_table, age_race_table])
table["total"] = table.sum(axis=1)

display(table)

print('Based on what I\'ve learned so far, if an algorithm is trained based on this dataset, the groups impacted the most \n' +
      'will be the minorities - ages 81-116 and individuals whose race is black - because any lessons learned from an \n' +
      'algorithm trained on this dataset will be applied to those minorities disproportionately. (The dispartiy between \n' +
      'males and females is not so significant to be a concern for being disproportinonately impacted by an algorithm \n' +
      'trained based on this dataset.) Moreoever, an exception to this might be any individuals who have been \n' +
      'miscategorized (especially if they were incorrectly placed into one the minority categories); for example, \n' +
      '21_0_4_20161223214826657.jpg.chip.jpg who is clearly not 21 years old at the time this photo was taken. Finding \n' +
      'duplicates may be fairly easy using a mean square error detection on each face, but finding miscategorizations \n' +
      'programmatically would be significantly trickier.')
```

	(0, 20]	(21, 40]	(41, 60]	(61, 80]	(81, 116]	total
female	2250	1518	711	423	223	5125
male	1889	847	881	462	107	4186
asian	1003	330	80	43	51	1507
black	157	96	75	52	14	394
indian	588	558	150	50	22	1368
other	524	380	81	9	2	996
white	1867	1001	1206	731	241	5046

```
Based on what I've learned so far, if an algorithm is trained based on this
dataset, the groups impacted the most
will be the minorities - ages 81-116 and individuals whose race is black -
because any lessons learned from an
algorithm trained on this dataset will be applied to those minorities
disproportionately. (The dispartiy between
males and females is not so significant to be a concern for being
disproportinonately impacted by an algorithm
trained based on this dataset.) Moreoever, an exception to this might be any
individuals who have been
miscategorized (especially if they were incorrectly placed into one the minority
categories); for example,
21_0_4_20161223214826657.jpg.chip.jpg who is clearly not 21 years old at the
time this photo was taken. Finding
```

```
duplicates may be fairly easy using a mean square error detection on each face,
but finding miscategorizations
programmatically would be significantly trickier.
```

[ ]:

```

```