This document is meant to help to understand the old simulation code.

1. Connect to Mongo database

2. Use Email Memory scheme

3. Initialize global variables:

sim\_round=100 corresponding to 100 subfolders in each result folder

sim\_round\_index=0 increments and goes back to 0 when reaches 100

links\_lower\_bound=1 how many link we are processing.

links\_upper\_bound=14 Currently we are using all 0-14

num\_emails=40 We have 40 emails originally in this study

duplicate\_level=3000 new added feature.

Initialize ACT-R Model parameters:

"subsymbolic": True, "rule\_firing": 0.05, "latency\_factor": 0.1,

"latency\_exponent": 1.0, "decay": 0.5, "baselevel\_learning": True,

"optimized\_learning": False, "instantaneous\_noise": 0.25,

"retrieval\_threshold": -float('inf'), "buffer\_spreading\_activation": {},

"spreading\_activation\_restricted": False, "strength\_of\_association": 0,

"association\_only\_from\_chunks": True, "partial\_matching": True,

"activation\_trace": True, "utility\_noise": 0, "utility\_learning": False,

"utility\_alpha": 0.2, "motor\_prepared": False, "strict\_harvesting": False,

"production\_compilation": False, "automatic\_visual\_search": True,

"emma": True, "emma\_noise": True, "emma\_landing\_site\_noise": False,

"eye\_mvt\_angle\_parameter": 1, "eye\_mvt\_scaling\_parameter": 0.01

4. ProcessByEmail()

This function processes the emails, with the help of ProcessByBatch and ProcessByCue using model.retrieval to retrieve cues. The function loops through all the emails and for each email, the function matches each cue to closest instance in the memory. The suspicious counter is raised when phishing cue is retrieved. If the suspicious counter reaches the configured suspicious threshold (2-7) before cue\_checked reaches the configured max\_cue (7-13), the simulation decision for the email is phishing.

Fault level is currently set from 0 to 0.6, increment by 0.1. A random flag value will decide whether or not a chunk is made properly.

If flag < fault level:

Make chunk

Higher fault level makes it harder to make a proper chunk.

Write out classification results and activation history.

**from** model **import** Email\_Cog\_Sim\_Model

**from** chunks **import** chunktype

**from** chunks **import** Chunk

**from** chunks **import** makechunk

**import** pymongo

**from** secrets **import** choice **as** rchoice

**import** pprint

**import** os

**from** shutil **import** copyfile

*# client = pymongo.MongoClient("192.168.129.129", 27017)#connect MongoDB Database*

client = pymongo.MongoClient(**"localhost"**, 27017)*#connect MongoDB Database*

*# Chunks = client['Memory\_Chunks']#use “Memory\_Chunks” scheme*

Chunks = client[**'Email\_Memory'**]*#use Email\_Memory scheme*

Updated\_Chunks = client[**'Updated\_Chunks'**]

collection = Updated\_Chunks.Emails

sim\_round = 100

sim\_round\_index = 0

links\_lower\_bound = 1

links\_upper\_bound = 14

num\_emails = 40

*# SUS\_THRESHOLD = 6*

*# CUE\_THRESHOLD = 11*

model\_params = {**"subsymbolic"**: **True**,

**"rule\_firing"**: 0.05,

**"latency\_factor"**: 0.1,

**"latency\_exponent"**: 1.0,

**"decay"**: 0.5,

**"baselevel\_learning"**: **True**,

**"optimized\_learning"**: **False**,

**"instantaneous\_noise"**: 0.25,

**"retrieval\_threshold"**: -float(**'inf'**),

**"buffer\_spreading\_activation"**: {},

**"spreading\_activation\_restricted"**: **False**,

**"strength\_of\_association"**: 0,

**"association\_only\_from\_chunks"**: **True**,

**"partial\_matching"**: **True**,

*# "mismatch\_penalty": 1.0,*

**"activation\_trace"**: **True**,

**"utility\_noise"**: 0,

**"utility\_learning"**: **False**,

**"utility\_alpha"**: 0.2,

**"motor\_prepared"**: **False**,

**"strict\_harvesting"**: **False**,

**"production\_compilation"**: **False**,

**"automatic\_visual\_search"**: **True**,

**"emma"**: **True**,

**"emma\_noise"**: **True**,

**"emma\_landing\_site\_noise"**: **False**,

**"eye\_mvt\_angle\_parameter"**: 1, *# in LispACT-R: 1*

**"eye\_mvt\_scaling\_parameter"**: 0.01, *# in LispACT-R: 0.01, but dft rule firing -- 0.01*

}

**def** ProcessByBatch(list,file,email):

**global** model\_params

retrieved\_chunk = **None**

used\_cue = **None**

activation\_score = float(**"-inf"**)

**global** current\_time

current\_time = current\_time + 1

file.write(**"Processing Batch: "** + list.\_\_repr\_\_() + **'\n'**)

**for** cue **in** list:

retrieved = retrieval.retrieve1(current\_time, cue, model\_params, dm, file, email)

**while** (retrieved[0] == **None**):

retrieved = retrieval.retrieve1(current\_time, cue, model\_params, dm, file, email)

**if** retrieved[1] > activation\_score:

retrieved\_chunk = retrieved[0]

used\_cue = cue

*# print("used\_cue", used\_cue)*

*# print("retrieved\_chunk",retrieved\_chunk)*

dm.add(retrieved\_chunk, current\_time)

file.write(**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**+**'\n'**)

file.write(**'Email: '** + str(email[**'Email'**]) + **"\n"**+**'Used Cue: '**+str(used\_cue.\_\_repr\_\_())+**'\n'**+**'Retrieved Chunk '** + str(retrieved\_chunk.\_\_repr\_\_())+**'\n'** + **'at current\_time: '** + str(current\_time) + **'\n'**)

**return** retrieved\_chunk,used\_cue

**def** ProcessByCue(cue,file,email):

**global** current\_time

**global** sim\_round\_index

**global** model\_params

*# a = list()*

*# p = list() # candidate*

*# for i in dm:*

*# a.append(i.match(cue, True))*

*#*

*# for j in dm:*

*# if j.match(cue, True) == max(a) and j.typename == cue.typename:*

*# p.append(j)*

current\_time = current\_time + 1.0

file.write(**"Processing Cue: "**+cue.\_\_repr\_\_()+**'\n'**)

**if** model\_params[**'activation\_trace'**]:

file.write(**"Activation Trace: "** + **'\n'**)

retrieved\_elem = retrieval.retrieve1(current\_time, cue, model\_params, dm,file,email)[0]

**while** (retrieved\_elem == **None**):

retrieved\_elem = retrieval.retrieve1(current\_time, cue,model\_params, dm,file,email)[0]

dm.add(retrieved\_elem, current\_time)

file.write(**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'**+**'\n'**)

file.write(**'Email: '** + str(email[**'Email'**]) + **"\n"**+**'Observed Cue: '**+cue.\_\_repr\_\_()+**'\n'**+**'Retrieved Chunk '** + retrieved\_elem.\_\_repr\_\_()+**'\n'** + **'at current\_time: '** + str(current\_time) + **'\n'**)

**return** retrieved\_elem

**def** ProcessByEmail(email,threshold,cue\_cutoff,file):

**global** current\_time

cue\_checked = 0

suspicion\_level = 0

sender\_chunk = makechunk(**"sender\_chunk"**, **"IDENTIFY\_SENDER\_NAME"**, email[**'Sender (Text)'**],sender=email[**'Sender'**])

subject\_chunk = makechunk(**"subject\_chunk"**, **"IDENTIFY\_EMAIL\_SUBJECT"**, email[**'Subject (Text)'**],subject=email[**'Subject'**])

branding\_chunk = makechunk(**"branding\_chunk"**, **"IDENTIFY\_BRANDING\_LOGOS"**,email[**'Branding (Text)'**], branding=email[**'Branding'**])

design\_chunk = makechunk(**"design\_chunk"**, **"IDENTIFY\_POOR\_DESIGN"**, email[**'Design (Text)'**],design=email[**'Design'**])

spelling\_chunk = makechunk(**"spelling\_chunk"**, **"IDENTIFY\_GRAMMAR\_ERROR"**, email[**'Spelling (Text)'**],spelling=email[**'Spelling'**])

greeting\_chunk = makechunk(**"greeting\_chunk"**, **"IDENTIFY\_GENERIC\_GREETING"**,email[**'Greeting (Text)'**], greeting=email[**'Greeting'**])

time\_chunk = makechunk(**"time\_chunk"**, **"IDENTIFY\_TIME\_PRESSURE"**, email[**'Time (Text)'**],time=email[**'Time'**])

threat\_chunk = makechunk(**"threat\_chunk"**, **"IDENTIFY\_THREAT\_LANG"**, email[**'Threats (Text)'**],threat=email[**'Threats'**])

emotion\_chunk = makechunk(**"emotion\_chunk"**, **"IDENTIFY\_EMOTION\_APPEAL"**, email[**'Emotion (Text)'**],emotion=email[**'Emotion'**])

signer\_chunk = makechunk(**"signer\_chunk"**, **"IDENTIFY\_SIGNER\_DETAIL"**, email[**'Signer (Text)'**],signer=email[**'Signer'**])

too\_good\_chunk = makechunk(**"too\_good\_chunk"**, **"IDENTIFY\_TOO\_GOOD"**, email[**'Toogood (Text)'**],tg=email[**'Toogood'**])

request\_chunk = makechunk(**"request\_chunk"**, **"IDENTIFY\_PERSONAL\_REQ"**, email[**'Requests (Text)'**],request=email[**'Requests'**])

random\_list = list()

linkchunk\_list = dict()

static\_list = list()

links\_count = 0

*#static cue list*

static\_list.append(branding\_chunk)

static\_list.append(design\_chunk)

static\_list.append(sender\_chunk)

static\_list.append(subject\_chunk)

static\_list.append(signer\_chunk)

static\_list.append(greeting\_chunk)

*#randomized cue list*

random\_list.append(time\_chunk)

random\_list.append(threat\_chunk)

random\_list.append(emotion\_chunk)

random\_list.append(too\_good\_chunk)

random\_list.append(request\_chunk)

random\_list.append(spelling\_chunk)

**for** i **in** range(links\_lower\_bound,links\_upper\_bound):

**if** email[**'Hyperlink (Text) #'**+str(i)] != **""**:

linkchunk\_list[**'link\_chunk'**+str(links\_count+1)] = makechunk(**"links\_chunk"**, **"IDENTIFY\_LINKS"**, email[**'Hyperlink (Text) #'**+str(i)],link\_in\_text=email[**'Hyperlink: Link\_In\_Text #'**+str(links\_count+1)],sus\_link=email[**'Hyperlink: Suspicious\_Link #'**+str(links\_count+1)])

links\_count = links\_count + 1

**else**:

**break**

**for** j **in** linkchunk\_list.keys():

random\_list.append(linkchunk\_list[j])

file.write(**'Opened Email: '**+str(email[**'Email'**])+**'\n'**)

**for** i **in** range(static\_list.\_\_len\_\_()):

cue = static\_list[i]

retrieved = ProcessByCue(cue,file,email)

cue\_checked = cue\_checked + 1

suspicion\_level = suspicion\_level + int(retrieved.criterion)

file.write(**'sus\_level: '** + str(suspicion\_level)+ **'\n'**)

file.write(**'cue\_checked: '** + str(cue\_checked) + **'\n'**)

file.write(

**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'** + **'\n'**)

**if** suspicion\_level >= threshold:

classification[email[**'Email'**]] = **"1"**

file.write(**'Current Time: '**+str(current\_time)+**' <Classified> Email: '** + **'<'** + str(

email[**'Email'**]) + **'>'** + **' to '** + **'<Suspicious>'** + **' at sus\_level '** + str(

suspicion\_level) + **" with "** + str(cue\_checked) + **" cues checked"** + **'\n'**)

file.write(

**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'** + **'\n'**)

**return**

**if** cue\_cutoff-len(static\_list) >0:

**for** i **in** range(cue\_cutoff-len(static\_list)):

retrieved\_entry = ProcessByBatch(random\_list, file, email)

retrieved = retrieved\_entry[0]

used = retrieved\_entry[1]

print(random\_list)

**for** i **in** random\_list:

**if** retrieved.typename == i.typename **and** retrieved.typename != **"IDENTIFY\_LINKS"** :

random\_list.remove(i)

**break**

**if** retrieved.typename == i.typename **and** retrieved.typename == **"IDENTIFY\_LINKS"**:

random\_list.remove(used)

**break**

print(random\_list.\_\_len\_\_())

cue\_checked = cue\_checked + 1

suspicion\_level = suspicion\_level + int(retrieved.criterion)

file.write(**'sus\_level: '** + str(suspicion\_level) + **'\n'**)

file.write(**'cue\_checked: '** + str(cue\_checked) + **'\n'**)

file.write(

**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'** + **'\n'**)

**if** suspicion\_level >= threshold:

classification[email[**'Email'**]] = **"1"**

file.write(**'Current Time: '**+str(current\_time)+**' <Classified> Email: '** + **'<'**+str(email[**'Email'**])+**'>'** + **' to '** + **'<Suspicious>'** + **' at sus\_level '** + str(suspicion\_level) + **" with "**+str(cue\_checked)+**" cues checked"**+**'\n'**)

file.write(

**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'** + **'\n'**)

**return**

*# cue= random\_list[randrange(0,len(random\_list),1)]*

*# retrieved = ProcessByCue(cue,file,email)*

*# suspicion\_level = suspicion\_level + int(retrieved.criterion)*

*# if suspicion\_level >= threshold:*

*# classification[email['Email']] = "1"*

*# file.write('email: ' + str(Email\_title) + ' sus\_level ' + str(suspicion\_level) + ' classification: ' + str(*

*# classification[email['Email']])+'\n')*

*# return*

*# random\_list.remove(cue)*

*# if len(linkchunk\_list.keys()) >=num\_links and len(linkchunk\_list.keys()) > 0:*

*# comb = combinations(range(len(linkchunk\_list.keys())), num\_links)*

*# for i in range(num\_links):*

*# cue = makechunk("link\_chunk", "IDENTIFY\_LINKS", link\_in\_text=email['Hyperlink: Link\_In\_Text #'+str(i+1)],sus\_link=email['Hyperlink: Suspicious\_Link #'+str(i+1)], criterion='')*

*# retrieved = ProcessByCue(cue,file,email)*

*# suspicion\_level = suspicion\_level + int(retrieved.criterion)*

*# if suspicion\_level >= threshold:*

*# classification[email['Email']] = "1"*

*# file.write('email: ' + str(Email\_title) + ' sus\_level ' + str(suspicion\_level) + ' classification: ' + str(*

*# classification[email['Email']])+'\n')*

*# return*

*# else:*

*# print(len(linkchunk\_list.keys()))*

*# raise ModelError("The email being processed do not contain as many links" )*

classification[email[**'Email'**]] = **"0"**

file.write(**'Current Time: '**+str(current\_time)+

**' <Classified> Email: '** + **'<'** + str(email[**'Email'**]) + **'>'** + **' to '** + **'<Keep>'** + **' at sus\_level '** + str(

suspicion\_level) + **" with "** + str(cue\_checked) + **" cues checked"** + **'\n'**)

file.write(

**'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'** + **'\n'**)

**return**

**if** \_\_name\_\_ == **'\_\_main\_\_'**:

**for** CUE\_THRESHOLD **in** range(7, 13):

**for** SUS\_THRESHOLD **in** range(2,7):

**for** mis\_coefficient **in** range(1,4):

model\_params[**'mismatch\_penalty'**] = float(mis\_coefficient)

*# folder\_path = str(CUE\_THRESHOLD)+'\_'+str(SUS\_THRESHOLD)+'\_'+str(mis\_coefficient)*

folder\_path = os.path.join(**''**, str(CUE\_THRESHOLD)+**'\_'**+str(SUS\_THRESHOLD)+**'\_'**+str(mis\_coefficient))

file\_path = os.path.join(folder\_path, **"accuracy"** + **"."** + **"py"**)

**if not** os.path.exists(folder\_path):

os.makedirs(folder\_path)

copyfile(**"accuracy.py"**, file\_path)

os.chdir(folder\_path)

**for** \_ **in** range(sim\_round):

model = Email\_Cog\_Sim\_Model()

dm = model.decmem

retrieval = model.retrieval

activation\_history = dm.activations

email\_list = list(range(num\_emails))

sim\_round\_index = (sim\_round\_index + 1)%100

file = open(**'results-'** + str(sim\_round\_index) + **'.txt'**, **'w'**, encoding=**'utf8'**)

current\_time = 0

classification = dict() *# Store classification result*

chunktype(**"IDENTIFY\_EMAIL\_SUBJECT"**, **"subject"**)

chunktype(**"IDENTIFY\_SENDER\_NAME"**, **"sender"**)

chunktype(**"IDENTIFY\_BRANDING\_LOGOS"**, **"branding"**)

chunktype(**"IDENTIFY\_POOR\_DESIGN"**, **"design"**)

chunktype(**"IDENTIFY\_GRAMMAR\_ERROR"**, **"spelling"**)

chunktype(**"IDENTIFY\_GENERIC\_GREETING"**, **"greeting"**)

chunktype(**"IDENTIFY\_TIME\_PRESSURE"**, **"time"**)

chunktype(**"IDENTIFY\_THREAT\_LANG"**, **"threat"**)

chunktype(**"IDENTIFY\_EMOTION\_APPEAL"**, **"emotion"**)

chunktype(**"IDENTIFY\_SIGNER\_DETAIL"**, **"signer"**)

chunktype(**"IDENTIFY\_TOO\_GOOD"**, **"tg"**)

chunktype(**"IDENTIFY\_PERSONAL\_REQ"**, **"request"**)

chunktype(**"IDENTIFY\_LINKS"**, (**"sus\_link"**, **"link\_in\_text"**))

subject = collection.find({},

{**"Subject (Text)"**: 1, **"Subject"**: 1, **"Criterion (Utility)"**: 1, **"\_id"**: 0})

sender = collection.find({}, {**"Sender (Text)"**: 1, **"Sender"**: 1, **"Criterion (Utility)"**: 1, **"\_id"**: 0})

branding = collection.find({}, {**"Branding (Text)"**: 1, **"Branding"**: 1, **"Criterion (Utility)"**: 1,

**"\_id"**: 0})

design = collection.find({}, {**"Design (Text)"**: 1, **"Design"**: 1, **"Criterion (Utility)"**: 1, **"\_id"**: 0})

spelling = collection.find({}, {**"Spelling (Text)"**: 1, **"Spelling"**: 1, **"Criterion (Utility)"**: 1,

**"\_id"**: 0})

greeting = collection.find({}, {**"Greeting (Text)"**: 1, **"Greeting"**: 1, **"Criterion (Utility)"**: 1,

**"\_id"**: 0})

time = collection.find({}, {**"Time (Text)"**: 1, **"Time"**: 1, **"Criterion (Utility)"**: 1, **"\_id"**: 0})

threats = collection.find({},

{**"Threats (Text)"**: 1, **"Threats"**: 1, **"Criterion (Utility)"**: 1, **"\_id"**: 0})

emotion = collection.find({},

{**"Emotion (Text)"**: 1, **"Emotion"**: 1, **"Criterion (Utility)"**: 1, **"\_id"**: 0})

signer = collection.find({}, {**"Signer (Text)"**: 1, **"Signer"**: 1, **"Criterion (Utility)"**: 1, **"\_id"**: 0})

toogood = collection.find({},

{**"Toogood (Text)"**: 1, **"Toogood"**: 1, **"Criterion (Utility)"**: 1, **"\_id"**: 0})

requests = collection.find({}, {**"Requests (Text)"**: 1, **"Requests"**: 1, **"Criterion (Utility)"**: 1,

**"\_id"**: 0})

links = dict()

**for** i **in** range(links\_lower\_bound, links\_upper\_bound):

links[**'link'** + str(i)] = collection.find({},

{**"Hyperlink (Text) #"** + str(i): 1,

**"Hyperlink: Link\_In\_Text #"** + str(i): 1,

**"Hyperlink: Suspicious\_Link #"** + str(i): 1,

**"Criterion (Utility)"**: 1,

**"\_id"**: 0})

**for** i **in** range(links\_lower\_bound, links\_upper\_bound):

**for** q **in** links[**'link'** + str(i)]:

**if** q[**'Hyperlink (Text) #'** + str(i)] != **""**:

dm.add(

makechunk(**"link\_chunk"**, **"IDENTIFY\_LINKS"**, str((q[**'Hyperlink (Text) #'** + str(i)])),

q[**'Criterion (Utility)'**],

link\_in\_text=q[**'Hyperlink: Link\_In\_Text #'** + str(i)],

sus\_link=q[**'Hyperlink: Suspicious\_Link #'** + str(i)]))

**for** i **in** subject:

dm.add(makechunk(**"subject\_chunk"**, **"IDENTIFY\_EMAIL\_SUBJECT"**, i[**'Subject (Text)'**],

i[**'Criterion (Utility)'**], subject=i[**'Subject'**]))

**for** i **in** sender:

dm.add(makechunk(**"sender\_chunk"**, **"IDENTIFY\_SENDER\_NAME"**, i[**'Sender (Text)'**],

i[**'Criterion (Utility)'**], sender=i[**'Sender'**]))

**for** i **in** branding:

dm.add(makechunk(**"branding\_chunk"**, **"IDENTIFY\_BRANDING\_LOGOS"**, i[**'Branding (Text)'**],

i[**'Criterion (Utility)'**], branding=i[**'Branding'**]))

**for** i **in** design:

dm.add(makechunk(**"design\_chunk"**, **"IDENTIFY\_POOR\_DESIGN"**, i[**'Design (Text)'**],

i[**'Criterion (Utility)'**], design=i[**'Design'**]))

**for** i **in** spelling:

dm.add(makechunk(**"spelling\_chunk"**, **"IDENTIFY\_GRAMMAR\_ERROR"**, i[**'Spelling (Text)'**],

i[**'Criterion (Utility)'**], spelling=i[**'Spelling'**]))

**for** i **in** greeting:

dm.add(makechunk(**"greeting\_chunk"**, **"IDENTIFY\_GENERIC\_GREETING"**, i[**'Greeting (Text)'**],

i[**'Criterion (Utility)'**], greeting=i[**'Greeting'**], ))

**for** i **in** time:

dm.add(makechunk(**"time\_chunk"**, **"IDENTIFY\_TIME\_PRESSURE"**, i[**'Time (Text)'**],

i[**'Criterion (Utility)'**], time=i[**'Time'**]))

**for** i **in** threats:

dm.add(makechunk(**"threat\_chunk"**, **"IDENTIFY\_THREAT\_LANG"**, i[**'Threats (Text)'**],

i[**'Criterion (Utility)'**], threat=i[**'Threats'**]))

**for** i **in** emotion:

dm.add(makechunk(**"emotion\_chunk"**, **"IDENTIFY\_EMOTION\_APPEAL"**, i[**'Emotion (Text)'**],

i[**'Criterion (Utility)'**], emotion=i[**'Emotion'**]))

**for** i **in** signer:

dm.add(makechunk(**"signer\_chunk"**, **"IDENTIFY\_SIGNER\_DETAIL"**, i[**'Signer (Text)'**],

i[**'Criterion (Utility)'**], signer=i[**'Signer'**]))

**for** i **in** toogood:

dm.add(makechunk(**"too\_good\_chunk"**, **"IDENTIFY\_TOO\_GOOD"**, i[**'Toogood (Text)'**],

i[**'Criterion (Utility)'**], tg=i[**'Toogood'**]))

**for** i **in** requests:

dm.add(makechunk(**"request\_chunk"**, **"IDENTIFY\_PERSONAL\_REQ"**, i[**'Requests (Text)'**],

i[**'Criterion (Utility)'**], request=i[**'Requests'**]))

**while** len(email\_list) > 0:

choice = rchoice(email\_list)

Email = collection.find().limit(-1).skip(choice).next()

Email\_title = collection.find().limit(-1).skip(choice).next()[**'Email'**]

ProcessByEmail(Email, SUS\_THRESHOLD, CUE\_THRESHOLD, file)

email\_list.remove(choice)

file.write(**"Classification Results"** + **"\n"**)

file.write(classification.\_\_repr\_\_() + **"\n"**)

file.write(**"Activation History: "** + **"\n"**)

*# file.write(dm.\_\_repr\_\_())*

pprint.pprint(dm.\_\_repr\_\_(), file)

file.close()

pprint.pprint(dm.\_\_repr\_\_())

os.system(**"python3 accuracy.py 1"**)

os.chdir(**".."**)