

**ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ**

**ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ: ΕΠΕΞΕΡΓΑΣΙΑ ΦΥΣΙΚΗΣ ΓΛΩΣΣΑΣ**

Επί των φοιτητή: Μόσχου Δημήτριου – Π18209

Διδάσκων: Παναγιωτόπουλος Θεμιστοκλής

Πειραιάς, 2021



**ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ**

**ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ: ΕΠΕΞΕΡΓΑΣΙΑ ΦΥΣΙΚΗΣ ΓΛΩΣΣΑΣ**

***Τελική Εργασία 2021***

Επί των φοιτητή: Μόσχου Δημήτριου – Π18209

Διδάσκων: Παναγιωτόπουλος Θεμιστοκλής

Πειραιάς, 2021

**ΠΡΟΛΟΓΟΣ 4**

**1. ΕΚΦΩΝΗΣΗ 5**

**2. ΘΕΜΑ 1 6**

**3. ΘΕΜΑ 2 12**

**4. ΘΕΜΑ 3 12**

**5. ΘΕΜΑ 4 12**

**ΠΡΟΛΟΓΟΣ**

Η παρούσα εργασία πραγματεύεται και ορίζει προς υλοποίηση τα παρακάτω θέματα:

• Ανάπτυξη & Τεκμηρίωση

• Λεκτικού Αναλυτή

• Συντακτικού Αναλυτή

• Σημασιολογικού Αναλυτή

• Διαχείριση Βάσης Γνώσης

Η εργασία αυτή, έχει υλοποιηθεί με την γλώσσα προγραμματισμού Python 3.9, σε συνδυασμό με το προγραμματιστικό εργαλείο PyCharm σε υπολογιστή MacBook Air 2019, με λογισμικό MacOS Big Sur.

ΘΕΜΑ 1.

Έχει χρησιμοποιηθεί η βιβλιοθήκη nltk, η οποία αφορά την Επεξεργασία Φυσικής Γλώσσας.

Η κύρια συνάρτηση της συγκεκριμένης εφαρμογής, η οποία και επιτυγχάνει την δημιουργία λιστών από προτάσεις, στην οποία η καθεμία περιέχει μια λίστα από λέξεις, είναι η sentence\_split().

Η συγκεκριμένη συνάρτηση έχει σαν όρισμα το κείμενο, το οποίο θα δεχτεί την παραπάνω διαδικασία.

Αρχικώς, για κάθε μια πρόταση θα εφαρμοστεί η μέθοδος tokenize(sent\_tokenize).

Στη συνέχεια για κάθε λέξη της πρότασης που βρίσκεται στην πρόταση, εφαρμόζεται και για αυτή η μέθοδος tokenize(word\_tokenize).

Οι λέξεις αυτές αποθηκεύονται σε μία λίστα, τη λίστα word.

Παρακάτω παρατίθεται, ο κώδικας της άσκησης:

**from** **nltk.tokenize** **import** word\_tokenize, sent\_tokenize

**def** **sent\_split**(documents):

words = [word\_tokenize(sent) **for** sent **in** sent\_tokenize(text)]

**return** words

text = 'My name is Dimitris and I am an Olympic Laser Sailing athlete from Greece !'

t=sent\_split(text)

print(t)

ΘΕΜΑ 2.

Έχει χρρησιμοποιηθεί η βιβλιοθήκη nltk, η οποία είναι η πλέον κατάλληλη βιβλιοθήκη της Επεξεργασία Φυσικής Γλώσσας στη Python.

Στο συγκεκριμένο πρόγραμμα, ανάλογα με την πρόταση την οποία έχουμε θα εμφανίζεται το αντίστοιχο συντακτικό δέντρο, το οποίο θα είναι σε μορφή functor.

Παρακάτω παρατίθεται, ο κώδικας της άσκησης:

**import** **nltk**

grammar = nltk.CFG.fromstring("""

S -> NP VP

NP -> PN | Det N | N

VP -> IV | IV Adv | AV Adj | TV PN NP | V NP

IV -> 'runs' | 'run' | 'running' | 'loves' | 'loved' | 'loving' | 'walks' | 'walk' | 'walking' | 'jumps' | 'jump' | 'jumping' | 'shoots' | 'shoot' | 'shooting'

AV -> 'is' | 'are' | 'does' | 'do'

TV -> 'plays' | 'play' | 'played' | 'playing'

PN -> 'kate' | 'james' | 'tomas'

Adv -> 'quickly' | 'slowly' | 'independently'

Det -> 'the' | 'a' | 'an'

N -> 'food' | 'cat' | 'dog' | 'dogs' | 'cat' | 'cats' | 'book' | 'books' | 'sport' | 'sports' | 'daddy' | 'daddies' | 'boy' | 'boys' | 'girls' | 'girl' | 'basketball' | 'basketballs'

Adj -> 'scary' | 'tall' | 'short' | 'blonde' | 'slim' | 'fat'

V -> 'chased' | 'chase' | 'needs' | 'hates' | 'hate' | 'has' | 'has' | 'have' | 'loves' | 'love' | 'kicks' | 'kick' | 'jumps' | 'jump'

""")

sentence = "kate played john a sport"

sent = sentence.split()

read\_parser = nltk.RecursiveDescentParser(grammar)

print("Parsing the sentence:"+ " " + sentence)

**for** tree **in** read\_parser.parse(sent):

print("The tree for the above sentence is:")

print(tree)

**break**

ΘΕΜΑ 3.

Παρ ’όλο που η φυσική γλώσσα μας είναι πεπερασμένη, οι σημασιολογικοί συνδυασμοί είναι άπειροι.

Είναι αδύνατον να αποθηκευτούν όλες οι προτάσεις ως έχουν, με αποτέλεσμα να χρησιμοποιούνται δυναμικά σημασιολογικά δίκτυα (semantic networks).

Για να μπορέσει να πραγματοποιηθεί η εκτέλεση του προγράμματος είναι απαραίτητο να έχουμε εγκαταστήσει τη βιβλιοθήκη gensim, όπως και τη βιβλιοθήκη ntlk, η οποία έχει χρησιμοποιηθεί και στα προηγούμενα θέματα .

Παρακάτω παρατίθεται, ο κώδικας της άσκησης:

#import modules

**import** **os.path**

**from** **gensim** **import** corpora

**from** **gensim.models** **import** LsiModel

**from** **nltk.tokenize** **import** RegexpTokenizer

**from** **nltk.corpus** **import** stopwords

**from** **nltk.stem.porter** **import** PorterStemmer

**from** **gensim.models.coherencemodel** **import** CoherenceModel

**import** **matplotlib.pyplot** **as** **plt**

**def** **load\_data**(path,file\_name):

"""

Input : path and file\_name

Purpose: loading text file

Output : list of paragraphs/documents and

title(initial 100 words considred as title of document)

"""

documents\_list = []

titles=[]

**with** open( os.path.join(path, file\_name) ,"r") **as** fin:

**for** line **in** fin.readlines():

text = line.strip()

documents\_list.append(text)

print("Total Number of Documents:",len(documents\_list))

titles.append( text[**0**:min(len(text),**100**)] )

**return** documents\_list,titles

**def** **preprocess\_data**(doc\_set):

"""

Input : docuemnt list

Purpose: preprocess text (tokenize, removing stopwords, and stemming)

Output : preprocessed text

"""

# initialize regex tokenizer

tokenizer = RegexpTokenizer(r'\w+')

# create English stop words list

en\_stop = set(stopwords.words('english'))

p\_stemmer = PorterStemmer()

texts = []

**for** i **in** doc\_set:

raw = i.lower()

tokens = tokenizer.tokenize(raw)

stopped\_tokens = [i **for** i **in** tokens **if** **not** i **in** en\_stop]

stemmed\_tokens = [p\_stemmer.stem(i) **for** i **in** stopped\_tokens]

texts.append(stemmed\_tokens)

**return** texts

**def** **prepare\_corpus**(doc\_clean):

"""

Input : clean document

Purpose: create term dictionary of our courpus and Converting list of documents (corpus) into Document Term Matrix

Output : term dictionary and Document Term Matrix

"""

dictionary = corpora.Dictionary(doc\_clean)

doc\_term\_matrix = [dictionary.doc2bow(doc) **for** doc **in** doc\_clean]

# generate LDA model

**return** dictionary,doc\_term\_matrix

**def** **create\_gensim\_lsa\_model**(doc\_clean,number\_of\_topics,words):

"""

Input : clean document, number of topics and number of words associated with each topic

Purpose: create LSA model using gensim

Output : return LSA model

"""

dictionary,doc\_term\_matrix=prepare\_corpus(doc\_clean)

# generate LSA model

lsamodel = LsiModel(doc\_term\_matrix, num\_topics=number\_of\_topics, id2word = dictionary) # train model

print(lsamodel.print\_topics(num\_topics=number\_of\_topics, num\_words=words))

**return** lsamodel

**def** **compute\_coherence\_values**(dictionary, doc\_term\_matrix, doc\_clean, stop, start=**2**, step=**3**):

"""

Input : dictionary : Gensim dictionary

corpus : Gensim corpus

texts : List of input texts

stop : Max num of topics

purpose : Compute c\_v coherence for various number of topics

Output : model\_list : List of LSA topic models

coherence\_values : Coherence values corresponding to the LDA model with respective number of topics

"""

coherence\_values = []

model\_list = []

**for** num\_topics **in** range(start, stop, step):

# generate LSA model

model = LsiModel(doc\_term\_matrix, num\_topics=number\_of\_topics, id2word = dictionary) # train model

model\_list.append(model)

coherencemodel = CoherenceModel(model=model, texts=doc\_clean, dictionary=dictionary, coherence='c\_v')

coherence\_values.append(coherencemodel.get\_coherence())

**return** model\_list, coherence\_values

**def** **plot\_graph**(doc\_clean,start, stop, step):

dictionary,doc\_term\_matrix=prepare\_corpus(doc\_clean)

model\_list, coherence\_values = compute\_coherence\_values(dictionary, doc\_term\_matrix,doc\_clean,

stop, start, step)

# Showσ graph

x = range(start, stop, step)

plt.plot(x, coherence\_values)

plt.xlabel("Number of Topics")

plt.ylabel("Coherence score")

plt.legend(("coherence\_values"), loc='best')

plt.show()

start,stop,step=**2**,**12**,**1**

plot\_graph(clean\_text,start,stop,step)

# LSA Model

number\_of\_topics=**7**

words=**10**

document\_list,titles=load\_data("","articles.txt")

clean\_text=preprocess\_data(document\_list)

model=create\_gensim\_lsa\_model(clean\_text,number\_of\_topics,words)

ΘΕΜΑ 4.

Στόχος της άσκησης είναι με βάση το αρχείο knowledge\_base.txt το οποίο δημιουργήθηκε να υπάρχει η δυνατότητα να δημιουργήσουμε ένα πρόγραμμα ερωταποκρίσεων , εισάγοντας σχέσεις στο txt αρχείο μας μέσω κάποιου αρχείου που δίνει ο χρήστης, εισάγοντας σχέσεις στο txt αρχείο μας μέσω ενός string (input) που δίνει ο χρήστης, έλεγχος ορθότητας μιας ερώτησης που κάνει ο χρήστης και διαγραφή βάση συγκεκριμένου index.

Για να να γίνει εφικτή η εκτέλεση του προγράμματος είναι απαραίτητο να έχουμε εγκατασταθούν οι βιβλιοθήκες spacy , pandas καθώς και η ntlk .

Παρακάτω παρατίθεται, ο κώδικας της άσκησης:

**import** **ast**

**import** **os**

**import** **nltk**

nltk.download('punkt')

**import** **pandas** **as** **pd**

**import** **spacy**

**from** **spacy.matcher** **import** Matcher

**import** **networkx** **as** **nx**

**import** **matplotlib.pyplot** **as** **plt**

nlp = spacy.load('en\_core\_web\_sm')

#Reads a string that contains that contains sentences and finds entities, relations and reloards

#.txt file of knowledge base

**def** **load\_sentences\_to\_knowledge\_base**(sstring,knlg\_database,relationsold):

sentences = nltk.tokenize.sent\_tokenize(sstring)

entities = []

relations = []

**for** i **in** range(len(sentences)):

entities.append(get\_entities(sentences[i]))

relations.append(get\_relation(sentences[i]))

knlg\_database = knlg\_database + entities

relationsold = relationsold + relations

open("knowledge\_base.txt", "w").close()

**with** open("knowledge\_base.txt", "w") **as** the\_file:

the\_file.write(str(knlg\_database) + "**\n**" + str(relationsold))

**return** knlg\_database, relationsold

#Reads a file and finds entities, relations and reloads

# .txt file of knoledge base

**def** **load\_file\_to\_knowledge\_base**(file\_name,knlg\_database,relationsold):

**try**:

text = read\_from\_file(file\_name)

sentences = nltk.tokenize.sent\_tokenize(text)

entities = []

relations = []

**for** i **in** range(len(sentences)):

entities.append(get\_entities(sentences[i]))

relations.append(get\_relation(sentences[i]))

knlg\_database = knlg\_database + entities

relationsold = relationsold + relations

open("knowledge\_base.txt", "w").close()

**with** open("knowledge\_base.txt", "w") **as** the\_file:

the\_file.write(str(knlg\_database)+"**\n**"+str(relationsold))

**return** knlg\_database,relationsold

**except** **OSError** **as** e:

print("File does not exist.")

**return** knlg\_database,relationsold

**def** **get\_entities**(sent):

sent = sent.lower()

ent1 = ""

ent2 = ""

prv\_tok\_dep = ""

prv\_tok\_text = ""

prefix = ""

modifier = ""

**for** token **in** nlp(sent):

#If token is a fullstop then go to next token

**if** token.dep\_ != "punct":

#Checks if token is a complex word

**if** token.dep\_ == "compound":

prefix = token.text

#If previous word was complex the add it

**if** prv\_tok\_dep == "compound":

prefix = prv\_tok\_text + " " + token.text

#If token is a modifier

**if** token.dep\_.endswith("mod") == **True**:

modifier = token.text

#If previous word was complex the add it

**if** prv\_tok\_dep == "compound":

modifier = prv\_tok\_text + " " + token.text

#If subject found

**if** token.dep\_.find("subj") == **True**:

entity1 = modifier + " " + prefix + " " + token.text

prefix = ""

modifier = ""

previous\_tok\_dep = ""

previous\_tok\_text = ""

#If object found

**if** token.dep\_.find("obj") == **True**:

entity2 = modifier + " " + prefix + " " + token.text

#Variable reload

prv\_tok\_dep = tok.dep\_

prv\_tok\_text = tok.text

print([ent1.strip(), ent2.strip()])

**return** [ent1.strip(), ent2.strip()]

**def** **get\_relation**(sent):

doc = nlp(sent)

# Matcher class object

matcher = Matcher(nlp.vocab)

#ορισμός του προτύπου

pattern = [{'DEP':'ROOT'},

{'DEP':'prep','OP':"?"},

{'DEP':'agent','OP':"?"},

{'POS':'ADJ','OP':"?"}]

matcher.add("matching\_1", **None**, pattern)

matches = matcher(doc)

k = len(matches) - **1**

span = doc[matches[k][**1**]:matches[k][**2**]]

print(span.text)

**return**(span.text)

#Διαβάζει ένα αρχείο και επιστρέφει το κείμενο που περιέχει

**def** **read\_from\_file**(file\_name):

file1 = open(file\_name, "r+", encoding="utf8")

print("The file text is:")

text = file1.read()

**return** text

#Plots know;edge graph

**def** **draw\_knowledge\_graph**(entities,relations):

# extract subject

source = [i[**0**] **for** i **in** entities]

# extract object

target = [i[**1**] **for** i **in** entities]

kg\_df = pd.DataFrame({'source': source, 'target': target, 'edge': relations})

# create a directed-graph from a dataframe

G = nx.from\_pandas\_edgelist(kg\_df, "source", "target",

edge\_attr=**True**, create\_using=nx.MultiDiGraph())

plt.figure(figsize=(**12**, **12**))

pos = nx.spring\_layout(G)

nx.draw(G, with\_labels=**True**, node\_color='skyblue', edge\_cmap=plt.cm.Blues, pos=pos)

plt.show()

#Converts list to a dictionary for every index of the list

**def** **list\_to\_dictionary\_with\_indexes**(lst):

dict = {}

**for** i **in** range(len(lst)):

dict[i] = lst[i]

**return** dict

#Διαγραφή ενός στοιχείου από την βάση σε κάποιο συγκεκριμένο index

**def** **delete\_from\_db**(index,entities,relations):

**try**:

entities.pop(index)

relations.pop(index)

open("knowledge\_base.txt", "w").close()

**with** open("knowledge\_base.txt", "w") **as** the\_file:

the\_file.write(str(entities) + "**\n**" + str(relations))

**return** entities, relations

**except** **IndexError**:

print("Not a valid index (out of range)")

**return** entities,relations

#Αν δεν υπάρχει αρχείο για την βάση γνώσης φτιάξε ένα

**if** **not** os.path.exists("knowledge\_base.txt"):

**with** open("knowledge\_base.txt", "w") **as** the\_file:

the\_file.write(str("[]**\n**[]"))

#Φόρτοση της βάσης γνώσης από το txt αρχείο στις ανάλογες λίστες

database = open("knowledge\_base.txt", 'r')

Lines = database.readlines()

knowledge\_database = ast.literal\_eval(Lines[**0**].strip())

relations\_database = ast.literal\_eval(Lines[**1**].strip())

draw\_knowledge\_graph(knowledge\_database, relations\_database)

**while** **True**:

# Menu with options

print('------------------------Menu------------------------')

print('1. Insert sentences from a file.')

print('2. Insert sentence/sentences as input')

print('3. Ask a question to the knowledge database')

print('4. Show the knowledge database')

print('5. Delete something from knowledge database')

print('6. Exit the script')

print("----------------------------------------------------")

**try**:

user\_choice = int(input("Choose:"))

#Αν δεν είναι ένας αριθμός από το 1-4 τότε η είσοδος είναι λάθος

**if** user\_choice<**0** **or** user\_choice>**6**:

print("Not a valid number. Choose a number from 1-4.")

**continue**

#Αν δεν είναι αριθμός τότε είναι λάθος

**except** **ValueError**:

print("Not a valid number. Choose a number from 1-4.")

#Αν ειναι η 6η επιλογή τερμάτισε το πρόγραμμα

**if** user\_choice == **6**:

exit(**0**)

#Αν είναι η 4η επιλογή τότε εμφάνισε την βάση σε μορφή πίνακα με ένα pandas dataframe

**if** user\_choice == **4**:

source = [i[**0**] **for** i **in** knowledge\_database]

target = [i[**1**] **for** i **in** knowledge\_database]

sdf = pd.DataFrame({'Entity[0]': source, 'Entity[1]': target, 'Relation': relations\_database})

print(sdf)

#Αν είναι η 1η επιλογή τότε φόρτωσε από το αρχείο που θα δοθεί από τον χρήστη

#στην βάση γνώσης

**if** user\_choice == **1**:

fl = input("File name:")

knowledge\_database,relations\_database = load\_file\_to\_knowledge\_base(fl,knowledge\_database,relations\_database)

draw\_knowledge\_graph(knowledge\_database, relations\_database)

#Αν είναι η 2η επιλογή τότε φόρτωσε από το string που θα δοθεί από τον χρήστη

#στην βάση γνώσης

**if** user\_choice == **2**:

snt = input("Write the sentence/sentences:")

knowledge\_database, relations\_database = load\_sentences\_to\_knowledge\_base(snt,knowledge\_database, relations\_database)

draw\_knowledge\_graph(knowledge\_database, relations\_database)

#Αν είναι η 3η επιλογή τότε ψάξε την ερώτηση που θα δώσει ο χρήστης στην βάση γνώσης και απάντησε

**if** user\_choice == **3**:

question = input("Write a question to the database:")

t = get\_entities(question)

**if** t **not** **in** knowledge\_database:

print("Output: No")

**else**:

r = knowledge\_database.index(t)

print("Output: ",relations\_database[r])

#Αν είναι η 5η επιλογή τότε γίνεται διαγραφή ενός στοιχείου από την βάση

**if** user\_choice == **5**:

print(list\_to\_dictionary\_with\_indexes(knowledge\_database))

print(list\_to\_dictionary\_with\_indexes(relations\_database))

**try**:

#Πάρε ως είσοδο το index για διαγραφή

indx = int(input("Choose a index from the dictionary to delete from the database:"))

knowledge\_database,relations\_database = delete\_from\_db(indx,knowledge\_database,relations\_database)

draw\_knowledge\_graph(knowledge\_database, relations\_database)

print("Deleted successfully")

**except** **ValueError**:

print("Not a valid index")

**continue**