

The tags in the LDA model :

Tag #1: that / with / from / this / which / also / their / have / they / such / research / health / were / other / culture / social / information / more / university / used

Tag #2: that / with / from / this / system / which / used / space / brain / also / have / such / water / when / chemical / method / other / more / these / been

Tag #3: equation / analysis / variance / rule / principle / data / image / carbon / component / multiple / play / measuring / average / examples / body / content / equations / mass / design / decision

Tag #4: displaystyle / equation / model / test / distribution / function / equations / that / probability / time / this / random / where / statistical / value / ratio / frac / number / formula / variable

Tag #5: mathematics / exercise / education / fitness / science / measurement / society / math / school / training / history / physical / algebra / psychology / course / biology / human / tree / weight / stem

Tag #6: theory / theorem / proof / equations / statistics / plant / index / mathematical / agricultural / agriculture / data / number / problem / algebraic / decision / effect / game / geometry / reliability / book

Tag #7: design / navigation / research / physician / diving / underwater / interaction / sound / electronic / choice / journal / natural / distance / nursing / foundation / video / methods / growth / culture / position

Tag #8: cultural / pollution / neuroscience / medical / laboratory / plastic / soil / cognitive / center / intelligence / framework / practice / board / perception / council / drug / development / information / neurons / media

Tag #9: chemistry / list / fishing / fish / show / international / network / chemical / industry / policy / measures / family / stage / change / complex / union / least / building / global / world

Tag #10: communication / visual / medicine / american / historical / practical / differences / service / five / equipment / experiment / experiences / civil / causes / field / time / velocity / properties / national / that

Obtained with this code version:

# -\*- coding: utf-8 -\*-

"""

Created on Wed Nov 13 00:23:32 2019

@author: theo local

"""

import time

import json

import numpy as np

import pandas as pd

from nltk import RegexpTokenizer

from nltk.corpus import stopwords

import gensim as sim

import re

from sklearn.cluster import DBSCAN

from sklearn.manifold import TSNE

import matplotlib.pyplot as plt

from gensim.models import Word2Vec

import wikipedia

import gensim

import tensorflow\_hub as hub

import tensorflow as tf

from rake\_nltk import Rake

from nltk.tokenize import word\_tokenize

import pickle

#from spacy.lang.es.stop\_words import STOP\_WORDS

#from stanfordcorenlp import StanfordCoreNLP

from nltk.stem import PorterStemmer, WordNetLemmatizer

from nltk.corpus import wordnet

import nltk

regex = RegexpTokenizer(r"\b\w+\b");

def processTranscript(transcript):

words = regex.tokenize(transcript)

processedTranscript = ""

for word in words:

word = word.lower()

processedTranscript = processedTranscript + word + " "

return processedTranscript

#Extract the common words from the transcripts and the pre trained word vectorizer

dataframe = pd.read\_excel(r"C:\Users\theod\Desktop\Erasmus\_Valencia\_2019-2020\Valencia\_work\ALEX\Valencia-Educ-Video\wikipediaArticles2.0.xlsx", parse\_cols = [1])

articles = dataframe.values

dataframe = pd.read\_excel(r"C:\Users\theod\Desktop\Erasmus\_Valencia\_2019-2020\Valencia\_work\ALEX\Valencia-Educ-Video\wikipediaArticles2.0.xlsx", parse\_cols = [2])

labels = dataframe.values

processedArticles = []

processedLabels = []

for i in range(0,len(articles)):

art = articles[i][0]

art = str(art)

if len(art) < 50:

continue

processedArticles.append(processTranscript(art))

processedLabels.append(labels[i])

from sklearn.model\_selection import train\_test\_split

wikiTrain, wikiEvalTrain, wikiTrainTest, wikiEvalTest = train\_test\_split(processedArticles,processedLabels , test\_size=0.30, random\_state=42)

wikiEvalTrain, wikiFinalTrain, wikiEvalTest, wikiFinalTest = train\_test\_split(wikiEvalTrain,wikiEvalTest , test\_size=0.50, random\_state=42)

print("Loading Transcripts")

transcripts = []

transcriptsKeywords = []

regex = RegexpTokenizer(r"\b\w+\b");

dictTranscriptTitle = {}

dictTranscriptTitleGabi = {}

dictTranscriptCluster = {}

dictTranscriptEmbed = {}

dictIndexTranscript = {}

dictIndexTranscriptGabi = {}

dictTitleCluster = {}

videoIdDictionary = {}

documentsGabi = []

counter = 0;

r = Rake(language="English")

i = 0

dataframe = pd.read\_excel(r"C:\Users\theod\Desktop\Erasmus\_Valencia\_2019-2020\Valencia\_work\ALEX\Valencia-Educ-Video\remake\_all\TaggedVideoTranscripts\_AllVideos.xlsx", parse\_cols = [2])

transcriptsData = dataframe.values

dataframe = pd.read\_excel(r"C:\Users\theod\Desktop\Erasmus\_Valencia\_2019-2020\Valencia\_work\ALEX\Valencia-Educ-Video\remake\_all\TaggedVideoTranscripts\_AllVideos.xlsx", parse\_cols = [3])

tags = dataframe.values

dataframe = pd.read\_excel(r"C:\Users\theod\Desktop\Erasmus\_Valencia\_2019-2020\Valencia\_work\ALEX\Valencia-Educ-Video\remake\_all\TaggedVideoTranscripts\_AllVideos.xlsx", parse\_cols = [1])

titles = dataframe.values

dictTranscriptTags = {}

dictTranscriptTitle = {}

index = 0

for transcript in transcriptsData:

str\_tags = re.sub(r'[^\w\s]','',tags[index][0])

tagsCollection = str\_tags.split(" ");

dictTranscriptTags[transcript[0]] = tagsCollection

dictTranscriptTitle[transcript[0]] = titles[index][0]

index += 1

r = Rake("Spanish")

for transcript in dictTranscriptTags:

#print(transcript)

keywords = dictTranscriptTags[transcript]

text\_keywords = ""

for word in keywords:

text\_keywords += word + " "

if transcript != "" and text\_keywords!= "":

transcripts.append(transcript)

dictIndexTranscript[counter] = transcript

#text\_keywords += dictTranscriptTitle[transcript]

transcriptsKeywords.append(text\_keywords)

counter = counter + 1

i = i + 1

print("Generating the vector matrix")

embed = hub.Module("https://tfhub.dev/google/nnlm-es-dim128-with-normalization/1")

X = []

Z = []

T = []

K = []

Q = []

with tf.Session() as session:

session.run([tf.global\_variables\_initializer(), tf.tables\_initializer()])

X = session.run(embed(wikiTrain))

Z = session.run(embed(wikiEvalTrain))

Q = session.run(embed(wikiFinalTrain))

T = session.run(embed(transcripts))

K = session.run(embed(transcriptsKeywords))

counter = 0

for i in range(0, len(T)):

dictTranscriptEmbed[dictIndexTranscript[counter]] = T[i]

counter = counter + 1

print(counter)

print("Vector matrix generated")

from sklearn import svm

from sklearn.model\_selection import cross\_val\_score

wikiTrainTest = np.array(wikiTrainTest)

from collections import defaultdict

validTranscriptsIndices = defaultdict(list)

wrongTranscriptsIndices = range(0,len(transcripts))

TranscriptsX = []

TranscriptsXLabels =[]

start = time.time()

while(True):

nr = 0

clf = svm.SVC(gamma=0.001, decision\_function\_shape='ovo', kernel='rbf', C=60)

#intre processed articles si labels

scores = cross\_val\_score(clf, X, wikiTrainTest.ravel(), cv=10)

print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))

#fit intre proccessed articles si labe;s

clf.fit(X, wikiTrainTest)

#prezice si pt articles de test

result = clf.predict(Z)

from sklearn.metrics import classification\_report

#cat de bine prezice label pt articles de test

print(classification\_report( (np.asarray(wikiEvalTest)).ravel() , result))

resultTrLabels = clf.predict(T)

resultKeyLabels = clf.predict(K)

validTranscriptsX = []

validTranscriptsLabels = []

invalidTranscriptsX = []

invalidTranscriptsLabels = []

invalidKeywordsX =[]

wrongIndices =[]

for i in range(0,len(resultTrLabels)):

if(resultTrLabels[i] == resultKeyLabels[i]):

validTranscriptsX.append(T[i])

for key\_transcript in dictTranscriptEmbed:

ok = 1

for k in range(0, len(dictTranscriptEmbed[key\_transcript])):

if(dictTranscriptEmbed[key\_transcript][k] != T[i][k]):

ok = 0

break

if(ok == 1):

dictTitleCluster[dictTranscriptTitle[key\_transcript]] = resultTrLabels[i]

TranscriptsX.append(T[i])

TranscriptsXLabels.append(resultTrLabels[i])

validTranscriptsLabels.append(resultTrLabels[i])

validTranscriptsIndices[resultTrLabels[i]].append(wrongTranscriptsIndices[i])

nr+=1

else:

invalidTranscriptsX.append(T[i])

invalidTranscriptsLabels.append(resultTrLabels[i])

invalidKeywordsX.append(K[i])

wrongIndices.append(wrongTranscriptsIndices[i])

wrongTranscriptsIndices = wrongIndices

print("Valid transcripts:" + str(nr) + " / " + str(len(resultTrLabels)) )

if(nr < 20):

for j in range(0,len(invalidTranscriptsX)):

TranscriptsX.append(invalidTranscriptsX[j])

TranscriptsXLabels.append(invalidTranscriptsLabels[j])

break

X = np.append(X, np.array(validTranscriptsX),axis = 0)

wikiTrainTest = np.append(wikiTrainTest, np.array(validTranscriptsLabels))

T = np.array(invalidTranscriptsX)

K = np.array(invalidKeywordsX)

if(nr == len(resultTrLabels)):

print(nr)

print(resultTrLabels)

break

file = open("newKeywordsComplete\_v2\_7clusters.txt", "w", encoding="utf-8")

clf = svm.SVC(gamma=0.001, decision\_function\_shape='ovo', kernel='rbf', C=60)

scores = cross\_val\_score(clf, np.array(TranscriptsX) ,np.array(TranscriptsXLabels).ravel(), cv=10)

print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std()))

clf.fit(np.array(TranscriptsX) ,np.array(TranscriptsXLabels).ravel())

result = clf.predict(Q)

end = time.time()

print("time: " + str(end - start))

print(classification\_report((np.asarray(wikiFinalTest)).ravel(), result))

titles = []

for title\_key in dictTitleCluster:

titles.append(title\_key)

titles.sort()

for title\_key in titles:

file.write(str(title\_key) + " " + str(dictTitleCluster[title\_key]))

file.write('\n')

file.close()

filename = 'alex\_modelRetrained\_hisWikiNewKeywords7CLusters.sav'

pickle.dump(clf, open(filename, 'wb'))