***BÁO CÁO BÀI TẬP LẬP TRÌNH 1***

Nguyễn Trung Sơn – 21025088

Lớp: 2223I\_INT6152

**Bài 1: Đưa ra 2 ví dụ độ tương tự cao, 2 ví dụ độ tương tự trung bình, 2 ví dụ độ tương tự thấp. Mỗi ví dụ là một cặp từ và giá trị tương tự cosine do chương trình của em tính được**

BL:

2 VD độ tương tự cao:

+ sim(khung\_thành, cầu\_môn) = 0.9057

+ sim(thuở, thủa)= 0.861

2 VD độ tương tự trung bình:

+ sim(ngày, đêm) = 0.5018

+ sim(hoang\_phí, hà\_tiện) = 0.5036

2 VD độ tương tự thấp:

+ sim(chỉnh\_hình, luồng) = - 0.1734

+ sim(đổ\_bê\_tông, biếu\_xén) = - 0.1728

**Code:**

def cosine\_similarity(u, v):

distance = 0.0

*# Compute the dot product between u and v (≈1 line)*

dot = np.dot(u,v)

*# Compute the L2 norm of u (≈1 line)*

norm\_u = np.sqrt(np.sum(u \* u))

*# Compute the L2 norm of v (≈1 line)*

norm\_v = np.sqrt(np.sum(v \* v))

*# Compute the cosine similarity defined by formula (1) (≈1 line)*

cosine\_similarity = dot / (norm\_u \* norm\_v)

return cosine\_similarity

file='DATASET/Visim-400.txt'

data\_split = pd.read\_csv(file, sep=" ")

words1 = []

words2 = []

for i in data\_split['Word1\tWord2\tPOS\tSim1\tSim2\tSTD']:

chunks = i.split('\t')

words1.append(chunks[0])

words2.append(chunks[1])

sims = []

for i\_w in range(len(words1)):

try:

s\_w1 = word\_to\_vec\_map[words1[i\_w]]

s\_w2 = word\_to\_vec\_map[words2[i\_w]]

sim = cosine\_similarity(s\_w1, s\_w2)

print(words1[i\_w],words2[i\_w])

print(sim)

sims.append(sim)

except KeyError:

pass

**Bài 2: Đưa ra 2 ví dụ, mỗi ví dụ gồm một từ được chọn và danh sách 10 từ (K=10) tương tự nhất với nó kèm với giá trị tương tự**

BL:

2 VD với k=10:

+ ('biến', [('loại\_bỏ', 0.39982202546609236), ('trừ\_khử', 0.3577853111161979), ('phát\_triển', 0.35468234283556915), ('cải\_biến', 0.3499645711349526), ('thuần\_hoá', 0.32730778322543275), ('tập\_hợp', 0.2970481096627491), ('đảo\_ngược', 0.29586987173526696), ('biến\_đổi', 0.2715603489232827), ('chuyển\_hướng', 0.25468541368467756), ('khơi\_gợi', 0.2513066830497953)])

+ ('nhà\_thi\_đấu', [('khán\_đài', 0.382081003073068), ('tập\_luyện', 0.3760700761773554), ('trận\_đấu', 0.3380625391209835), ('sân\_khấu', 0.2549420655275232), ('trận\_chiến', 0.22814374864687503), ('tiểu\_học', 0.21454221562473896), ('giở\_chứng', 0.18710773491409094), ('đêm', 0.18002264612377802), ('công\_trình\_phụ', 0.1772424345078385), ('bảng\_tổng\_sắp', 0.17701107414308503)])

**Code:**

def k\_nearest\_words(word, all\_words):

all\_sims = []

top\_k = dict()

try:

word\_vector = word\_to\_vec\_map[word]

for idx in all\_words:

try:

idx\_vector = word\_to\_vec\_map[idx]

similarity = cosine\_similarity(word\_vector, idx\_vector)

top\_k.update({idx:similarity})

except KeyError:

pass

return (word, sorted(top\_k.items(), key=lambda x: x[1], reverse=True)[:10])

except KeyError:

pass

for e\_w in words1:

print(k\_nearest\_words(e\_w, words2),'\n')

**Bài 3:**

**- Nêu tên mô hình phân lớp mà em chọn (Logistic Regression, Multi-layer Perceptron, …), lý do lựa chọn.**

**- Tên thư viện hoặc mã nguồn mở học máy mà em sử dụng (hoặc tự code thì nói rõ), kèm theo link.**

**- Kết quả đánh giá tính theo Precision, Recall, F1**

**- Ví dụ về 2 trường hợp phân lớp sai và giải thích**

**- Đoạn code em đã lập trình.**

BL:

* Mô hình em chọn là **Multi-layer Perceptron.** Vì dữ liệu là dữ liệu có cấu trúc và vì thế trong trường hợp này nên sử dụng deep learning thay vì machine learning cổ điển.
* Trong trường hợp này em tự one hot encoding và pair 2 words vào với nhau thành một cặp sau đó train với tỷ lệ (train, test)=(80,20), cuối cùng em lấy data train với [MLPClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html) của thư viện sklearn.
* **Precision** = 98,3%, **Recall** = 97%, **F1** = 97.65%
* 2 TH phân lớp sai:
  + vững vững\_vàng ANT : Có thể model phân vân giữa chữ một ngữ nghĩa và 2 ngữ nghĩa và đoán sai
  + tối\_tăm tối\_dạ ANT : Trong tập trai có thể có nhiều cặp khác nghĩa những đồng âm với 2 từ này và các cặp đó đều SYN nên dẫn tới predict cặp này sai.

**Code:**

ant = open('DATASET/Antonym\_vietnamese.txt', 'r').read().strip('\n').split('\n')

syn = open('DATASET/Synonym\_vietnamese.txt', 'r').read().strip('\n').split('\n')

corpus = []

for pair in ant:

corpus.extend(pair.split(' '))

for pair in syn:

corpus.extend(pair.split(' '))

corpus = list(set(corpus))

corpus = {word: i for (i, word) in enumerate(corpus)}

corpus['UNK'] = len(corpus)

X = []

y = []

print(len(corpus))

for pair in ant:

pair = pair.split(' ')

if len(pair) == 2: *# contain 2 words*

w1, w2 = pair[0], pair[1]

*# one-hot encoding*

one\_hot1 = np.zeros(len(corpus))

one\_hot1[corpus[w1]] = 1

one\_hot2 = np.zeros(len(corpus))

one\_hot2[corpus[w2]] = 1

X.append([one\_hot1, one\_hot2])

y.append(0)

for pair in syn:

pair = pair.split(' ')

if len(pair) == 2: *# contain 2 words*

w1, w2 = pair[0], pair[1]

one\_hot1 = np.zeros(len(corpus))

one\_hot1[corpus[w1]] = 1

one\_hot2 = np.zeros(len(corpus))

one\_hot2[corpus[w2]] = 1

X.append([one\_hot1, one\_hot2])

y.append(1)

X = np.array(X)

y = np.array(y)

X = X.reshape(X.shape[0], -1)

print(X.shape, y.shape)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

epochs = 100

clf = MLPClassifier(hidden\_layer\_sizes=(256, 100, 100, 50), random\_state=1, max\_iter=epochs, verbose=True).fit(X\_train, y\_train)

pred = clf.predict(X\_val)

score = f1\_score(pred, y\_val)

print("F1 for validate dataset is: ", score)

*#TEST WITH ViCon-400 DATASET*

nouns ='DATASET/ViCon-400/400\_noun\_pairs.txt'

verbs = 'DATASET/ViCon-400/400\_verb\_pairs.txt'

adjs = 'DATASET/ViCon-400/600\_adj\_pairs.txt'

nouns\_split = pd.read\_csv(nouns, sep=" ")

verbs\_split = pd.read\_csv(verbs, sep=" ")

adjs\_split = pd.read\_csv(adjs, sep=" ")

all\_words = []

labels = []

X\_test, y\_test = [], []

*#Concat three dataset to one*

for i in nouns\_split['Word1\tWord2\tRelation']:

w1, w2, label = i.split('\t')

one\_hot1 = np.zeros(len(corpus))

if corpus.get(w1) is None:

w1 = 'UNK'

if corpus.get(w2) is None:

w2 = 'UNK'

one\_hot1[corpus[w1]] = 1

one\_hot2 = np.zeros(len(corpus))

one\_hot2[corpus[w2]] = 1

X\_test.append([one\_hot1, one\_hot2])

label = 1 if label=='SYN' else 0

y\_test.append(label)

for i in verbs\_split['Word1\tWord2\tRelation']:

w1, w2, label = i.split('\t')

one\_hot1 = np.zeros(len(corpus))

if corpus.get(w1) is None:

w1 = 'UNK'

if corpus.get(w2) is None:

w2 = 'UNK'

one\_hot1[corpus[w1]] = 1

one\_hot2 = np.zeros(len(corpus))

one\_hot2[corpus[w2]] = 1

X\_test.append([one\_hot1, one\_hot2])

label = 1 if label=='SYN' else 0

y\_test.append(label)

for i in adjs\_split['Word1\tWord2\tRelation']:

w1, w2, label = i.split('\t')

one\_hot1 = np.zeros(len(corpus))

if corpus.get(w1) is None:

w1 = 'UNK'

if corpus.get(w2) is None:

w2 = 'UNK'

one\_hot1[corpus[w1]] = 1

one\_hot2 = np.zeros(len(corpus))

one\_hot2[corpus[w2]] = 1

X\_test.append([one\_hot1, one\_hot2])

label = 1 if label=='SYN' else 0

y\_test.append(label)

X\_test = np.array(X\_test)

y\_test = np.array(y\_test)

X\_test = X\_test.reshape(X\_test.shape[0], -1)

pred = clf.predict(X\_test)

f1\_score = f1\_score(pred, y\_test)

recall = recall\_score(pred, y\_test)

precision = precision\_score(pred, y\_test)

print("The F1-score of Vicon-400 dataset is: "+ str(f1\_score\*100) + "%")

print("Recall is:", recall)

print("Precision is: ", precision)