NLP HW4

jha2137

April 2020

1 Problem 1

EUCLIDEAN(DOG, ANIMAL) =

$$[(0-2)^2+(4-3)^2+(0-0)^2+(4-3)^2+(2-0)^2+(2-3)^2]^{\frac{1}{2}}=\sqrt{11}$$

COS(DOG, ANIMAL) =

$$\frac{0+12+0+12+0+6}{\sqrt{40}\sqrt{31}}=.851$$

EUCLIDEAN(CAT, ANIMAL) =

$$[(4-2)^2+(0-3)^2+(0-0)^2+(3-3)^2+(3-0)^2+(10-3)^2]^{\frac{1}{2}}=\sqrt{134}$$

COS(CAT, ANIMAL) =

$$\frac{8+0+0+9+0+30}{\sqrt{134}\sqrt{31}}=.729$$

EUCLIDEAN(COMPUTER, ANIMAL) =

$$[(0-2)^2+(0-3)^2+(0-0)^2+(5-3)^2+(0-0)^2+(5-3)^2]^{\frac{1}{2}}=\sqrt{50}$$

COS(COMPUTER, ANIMAL) =

$$\frac{0+0+0+15+0+15}{\sqrt{50}\sqrt{31}}=.762$$

EUCLIDEAN(RUN, ANIMAL) =

$$[(4-2)^2+(3-3)^2+(5-0)^2+(0-3)^2+(3-0)^2+(4-3)^2]^{\frac{1}{2}}=\sqrt{75}$$

COS(RUN, ANIMAL) =

$$\frac{8+9+0+0+0+12}{\sqrt{75}\sqrt{31}}=.601$$

EUCLIDEAN(MOUSE, ANIMAL) =

$$[(2-2)^2 + (10-3)^2 + (5-0)^2 + (4-3)^2 + (3-0)^2 + (0-3)^2]^{\frac{1}{2}} = \sqrt{11}$$

COS(MOUSE, ANIMAL) =

$$\frac{4 + 30 + 0 + 12 + 0 + 0}{\sqrt{153}\sqrt{31}} = .667$$

Dog is the closest to animal according to euclidean distance and it is the most similar to animal according to cos similarity. Therefore, we conclude that dog is the most similar word to animal out of the options presented.

2 Problem 2

The way we will parse this using wordnet is to compare the similarity of the examples presented in wordnet. We will vectorize the examples, and then compare the similarity between them using cos similarity. If the examples are similar such that it is greater than some threshold α then we will say they are polysemy. If the similarity is less than α we will label them as homonyms.

3 Code

```
#!/usr/bin/env python
import sys
import time
from lexsub_xml import read_lexsub_xml

# suggested imports
from nltk.corpus import wordnet as wn
from nltk.corpus import stopwords
import gensim
import string
import numpy as np
from collections import defaultdict
# Participate in the 4705 lexical substitution competition (optional): NO
# Alias: [please invent some name]

def tokenize(s):
    s = "".join(" " if x in string.punctuation else x for x in s.lower())
    return s.split()

def get_candidates(lemma, pos):
    # Part 1
    possible_synonyms = []
    l1 = wn.lemmas(lemma, pos=pos)
    for lexeme in l1:
        syn = lexeme.synset().lemmas()
        for i in range(len(syn)):
            if syn[i].name() in possible_synonyms or syn[i].name() == lemma:
                continue
            elif "_" in syn[i].name():
                possible_synonyms.append(" ".join(syn[i].name().split("_")))
            else:
                possible_synonyms.append(syn[i].name())
    return possible_synonyms

def smurf_predictor(context):
    """
    Just suggest 'smurf' as a substitute for all words.
    """
    return 'smurf'

def wn_frequency_predictor(context):
    l = wn.lemmas(context.lemma, pos = context.pos)
    pairs = defaultdict(int)
    for lexeme in l:
        syn = lexeme.synset().lemmas()
        for i in syn:
            if i.name() == context.lemma:
                continue
            pairs[i.name()] += i.count()
    return max(pairs.items(), key = lambda a: a[1])[0]

def wn_simple_lesk_predictor(context):
    l = wn.lemmas(context.lemma, pos = context.pos)
    stop_words = stopwords.words('english')
    stop_words = set(stop_words)
    max_length = 0
    best = None
    for lexeme in l:
        syn = lexeme.synset()
        defin = set(tokenize(syn.definition()))
        for i in syn.examples():
            defin = defin.union(set(tokenize(i)))
        for j in syn.hypernyms():
            defin = defin.union(set(tokenize(j.definition())))
            for k in j.examples():
                defin = defin.union(set(tokenize(k)))
        defin -= stop_words
        contextset = set(tokenize(str(context)))
        contextset -= stop_words
        intersect = defin.intersection(contextset)
        if len(intersect) > max_length:
            best = syn
            max_length = len(intersect)
    if best is None:
        return wn_frequency_predictor(context)
    poss = {}
    for word in best.lemmas():
        if word.name() == context.lemma:
            continue
        else:
            poss[word.name()] = word.count()
    if not bool(poss) or set(poss.values()) == set([0]):
        return wn_frequency_predictor(context)
    return max(poss.items(), key = lambda a: a[1])[0]

def cos(v1,v2):
        return np.dot(v1,v2) / (np.linalg.norm(v1)*np.linalg.norm(v2))

class Word2VecSubst(object):
```

```
def __init__(self, filename):
    self.model = gensim.models.KeyedVectors.load_word2vec_format(filename, binary=Tr

def predict_nearest(self,context):
    pos_syn = get_candidates(context.lemma, context.pos)
    sim = 0
    guess = None
    guesses = {}
    for syn in pos_syn:
        try:
            guesses[syn] = self.model.similarity(syn, context.lemma)
            if self.model.similarity(syn, context.lemma) > sim:
                guess = syn
                sim = self.model.similarity(syn, context.lemma)
        except KeyError:
            continue
    print("NO CONTEXT\n",guesses)
    return guess


def predict_nearest_with_context(self, context):
    vec = np.array(self.model.wv[context.lemma])
    count = 0
    revLcont = context.left_context
    revLcont.reverse()
    Rcont = context.right_context
    stop_words = stopwords.words('english')
    stop_words = set(stop_words)
    pos_syn = get_candidates(context.lemma, context.pos)
    for i in revLcont:
        if i in stop_words or i in string.punctuation:
            continue
        elif count > 4:
            break
        else:
            try:
                vec += np.array(self.model.wv[i])
                count += 1
            except KeyError:
                continue
    count = 0
    for j in Rcont:
        if j in stop_words or j in string.punctuation:
            continue
        elif count > 4:
            break
        else:
            try:
                vec += np.array(self.model.wv[j])
                count += 1
            except KeyError:
                continue
    sim = 0
    for syn in pos_syn:
        try:
            if cos(self.model.wv[syn], vec) > sim:
                guess = syn
                sim = cos(self.model.wv[syn], vec)
        except KeyError:
            continue
    return guess

def new_predict(self, context): # Add a snapier name after completion
    vec = np.array(self.model.wv[context.lemma])
    count = 0
    revLcont = context.left_context
    revLcont.reverse()
    Rcont = context.right_context
    stop_words = stopwords.words('english')
    stop_words = set(stop_words)
    pos_syn = get_candidates(context.lemma, context.pos)
    for i in revLcont:
        if i in stop_words or i in string.punctuation:
            continue
        elif count > 4:
            break
        else:
            try:
                vec += np.array(self.model.wv[i])
                count += 1
            except KeyError:
                continue
    count = 0
    for j in Rcont:
        if j in stop_words or j in string.punctuation:
            continue
        elif count > 4:
            break
        else:
            try:
                vec += np.array(self.model.wv[j])
                count += 1
            except KeyError:
                continue
        guesses = {}
        for syn in pos_syn:
            try:
                guesses[syn] = cos(self.model.wv[syn], vec)
                guesses[syn] += self.model.similarity(syn, context.lemma)
            except KeyError:
                continue

        return max(guesses.items(), key = lambda a: a[1])[0]


if __name__=="__main__":
# At submission time, this program should run your best predictor (part 6).
start_time = time.time()
W2VMODEL_FILENAME = 'GoogleNews-vectors-negative300.bin.gz'
predictor = Word2VecSubst(W2VMODEL_FILENAME)
filename = "freq.predict"
file = open(filename, "w")
count = 0

print("STARTING", time.time() - start_time)
for context in read_lexsub_xml(sys.argv[1]):
    prediction = predictor.new_predict(context)

    count += 1
    file.write("{}.{} {} :: {}".format(context.lemma, context.pos, context.cid, predi
    file.write("\n")

file.close()
```