import csv

import operator

from nltk import word\_tokenize

from nltk import stem

import os

from dataManager import DataManager

from nltk.tag.stanford import CoreNLPPOSTagger, CoreNLPNERTagger

from nltk.tokenize.stanford import CoreNLPTokenizer

from nltk.tag.stanford import StanfordPOSTagger

import nltk

import pandas as pd

java\_path = "C:\Program Files\Java\jdk1.8.0\_151" # replace this

os.environ['JAVA\_HOME'] = java\_path

import re

class SubLexicon:

def \_\_init\_\_(self, word, pos, isStem, polarity):

self.\_\_word = word

self.\_\_pos = self.get\_pos(pos)

self.\_\_stemmed = \

True if isStem == 'y' else False # True if stemmed=y, else False

self.\_\_polarity = polarity

def get\_pos(self, pos):

if pos == 'noun':

return ['NN', 'NNS', 'NNP', 'NNPS']

elif pos == 'adj':

return ['JJ', 'JJR', 'JJS']

elif pos == 'verb':

return ['VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ']

else:

return ['ANY']

def get\_polarity(self):

return self.\_\_polarity

class POS:

nouns = []

adjs = []

verbs = []

features\_3\_pos = []

features\_all\_words = []

subjectivity\_lexicons = {}

def extract\_bow\_3\_pos\_tags(self, filename):

"""

2. a) Extract a bag-of-words list of nouns, adjectives, and verbs for all targets individually

:param filename:

:return:

"""

with open(filename, 'r') as csvfile:

reader = csv.reader(csvfile, delimiter=',')

for row in reader:

# print(row)

if row[1] in ['NN', 'NNS', 'NNP']:

self.nouns.append(row[0])

elif row[1] in ['VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ']:

self.verbs.append(row[0])

elif row[1] in ['JJ', 'JJR', 'JJS']:

self.adjs.append(row[0])

self.features\_3\_pos.extend(self.nouns)

self.features\_3\_pos.extend(self.verbs)

self.features\_3\_pos.extend(self.adjs)

# remove duplicate features

self.features\_3\_pos = list(set(self.features\_3\_pos))

#print("features (after duplicates removed) are {}".format(self.features\_3\_pos.\_\_len\_\_()))

def extract\_bow\_all\_words(self, filename):

"""

2. a) Extract a bag-of-words list of all words for all targets individually

:param filename:

:return:

"""

with open(filename, 'r') as csvfile:

reader = csv.reader(csvfile, delimiter=',')

for row in reader:

# print(row)

self.features\_all\_words.append(row[0])

# remove duplicate features

self.features\_all\_words = list(set(self.features\_all\_words))

#print("features (after duplicates removed) are {}".format(self.features\_all\_words.\_\_len\_\_()))

def create\_file(self, filename, new\_filename, all\_words=False):

"""

Use those words as features and create a file in which the feature values are either 1 or 0 depending

on whether the corresponding word is in the tweet or not. Add the tweet label as the last element

(gold class) in every line.

:param filename:

:return:

"""

with open(new\_filename, 'w') as wp, open(filename, 'rU') as rp:

writer = csv.writer(wp, delimiter=',')

reader = csv.reader(rp, delimiter=',')

# if not all\_words:

# writer.writerow(self.features\_3\_pos)

# else:

# writer.writerow(self.features\_all\_words)

# f = open("test/trump\_test.csv")

# for row in f.readlines():

# print row

n = 30

for row in reader:

#print row

r = []

nrow = []

if len(row)>2:

nrow.append(row[:-2])

nrow.append(row[-2])

nrow.append(row[-1])

#text = word\_tokenize(row[0])

pattern = r"'([A-Za-z0-9\_\./\\-]\*)'"

m = re.findall(pattern, str(nrow[0]))

nrow[0] = m

#print row[0], row[1], row[2]

if not all\_words:

for feature in nrow[0]:

#print feature, row[0], feature in row[0]

if feature in self.features\_3\_pos:

r.append('1')

else:

r.append('0')

else:

for feature in nrow[0]:

if feature in self.features\_all\_words:

r.append('1')

else:

r.append('0')

rem = n - len(r)

#print n, len(r), rem

for l in range(rem):

r.append('0')

try:

#print r

#print nrow[-1], nrow

r = r[:n]

mylist = ["FAVOR", "AGAINST", "NONE"]

if(nrow[-1] in mylist):

r.append(nrow[-1]) # stance column

writer.writerow(r)

writer.write("\n")

except:

pass

def read\_subjectivity\_lexicons(self, sublex\_filename):

"""

Reads the subjectivity lexicon file, and constructs the datastructure,

finally adds to the 'subjectivity\_lexicons' list

:param sublex\_filename:

:return:

"""

with open(sublex\_filename) as rp:

for row in rp:

line\_words = word\_tokenize(row)

# print line\_words

# parse the line

lexicon\_word = line\_words[2].split('=')[1]

lexicon\_pos = line\_words[3].split('=')[1]

lexicon\_stemmed = line\_words[4].split('=')[1]

lexicon\_polarity = line\_words[5].split('=')[1]

# create new DS object & add to list

self.subjectivity\_lexicons[lexicon\_word] = SubLexicon(word=lexicon\_word,

pos=lexicon\_pos,

isStem=lexicon\_stemmed,

polarity=lexicon\_polarity)

print ("total num. subjectivity lexicons = {}".format(self.subjectivity\_lexicons.\_\_len\_\_()))

print ("test lexicon polarity = {}".format(self.subjectivity\_lexicons['abandoned'].get\_polarity()))

pass

def create\_features\_with\_sublex(self, filename, new\_filename):

with open(new\_filename, 'w') as wp, open(filename, 'rU') as rp:

writer = csv.writer(wp, delimiter=',')

reader = csv.reader(rp, delimiter=',')

# print (self.features\_all\_words)

# create stemmer for extracting stems of words

stemmer = stem.PorterStemmer()

for row in reader:

print row

r = []

text = word\_tokenize(row[0])

for feature in self.features\_all\_words:

if feature in text:

# check if feature or the stem of the feature

# is in subjectivity lexicon

feature\_stem = stemmer.stem(feature)

feature = feature.lower()

if feature\_stem in self.subjectivity\_lexicons:

lexicon\_obj = self.subjectivity\_lexicons[feature\_stem]

if lexicon\_obj.get\_polarity() == 'positive':

r.append('1')

else:

r.append('-1')

elif feature in self.subjectivity\_lexicons:

lexicon\_obj = self.subjectivity\_lexicons[feature]

if lexicon\_obj.get\_polarity() == 'positive':

r.append('1')

else:

r.append('-1')

else:

r.append('0')

else:

r.append('0')

r.append(row[2]) # stance column

writer.writerow(r)

pass

#Tweet,Target,Stance,Opinion Towards,Sentiment

def calculate\_baseline(self, train\_filename, test\_filename):

with open(train\_filename, 'rU') as rp, open(test\_filename, 'rU') as tp:

reader\_train = csv.reader(rp, delimiter=',')

reader\_test = csv.reader(tp, delimiter=',')

# num. of examples in test

test\_data = list(reader\_test)

total\_test\_size = len(test\_data)

# count number of classes

classes = {'FAVOR': 0, 'AGAINST': 0, 'NONE': 0}

for row in reader\_train:

#print(row)

classes[row[-1]] += 1 #row[2]

max\_class = max(classes.iteritems(), key=operator.itemgetter(1))[0]

print ("max class for this data-set is: {}".format(max\_class))

misclassification\_count = 0

for row in test\_data:

if not row[-1] == max\_class: #row[2]

misclassification\_count += 1

baseline\_accuracy = float(total\_test\_size - misclassification\_count) / float(total\_test\_size)

print ("data-set size = {}, misclassification count = {}. Hence baseline accuracy = {}".format(total\_test\_size, misclassification\_count, baseline\_accuracy))

pass

def makeDirs(directory):

if not os.path.exists(directory):

os.makedirs(directory)

def pos\_tag(series):

# def rem\_mentions\_hasht(tweet):

# words = tweet.split()

# relevant\_tokens = [w for w in words if '@' not in w and '#' not in w]

# return( " ".join(relevant\_tokens))

words = list(series[0])

#tokens = [w for w in words if '@' not in w and '#' not in w]

stopSymbols = ["@", "#"]

for word in range(len(words)):

if words[word][0] in stopSymbols:

#print words[word][0]

words[word] = words[word][1:]

#series = series.apply(lambda tweet: rem\_mentions\_hasht(tweet))

#series = [rem\_mentions\_hasht(series[0]).split()]

#print ("done 1")

# from nltk.tag.stanford import StanfordPOSTagger

# import os

# java\_path = "C:/Program Files/Java/jre1.8.0\_111/bin/java.exe"

# os.environ['JAVAHOME'] = java\_path

#coreNLPTokenizer = CoreNLPTokenizer()

english\_postagger = StanfordPOSTagger(java\_options = "-mx7g", model\_filename='stanford\_models/english-bidirectional-distsim.tagger', path\_to\_jar='stanford-postagger-3.8.0.jar')

#from nltk.internals import find\_jars\_within\_path

#from nltk.parse.stanford import StanfordParser

#english\_postagger = StanfordParser(model\_path="stanford\_models/englishPCFG.ser.gz")

#english\_postagger.\_classpath = tuple(find\_jars\_within\_path('.'))

# #StanfordPOSTagger('stanford-postagger-3.8.0.jar')

#print (nltk.pos\_tag(words))

x = english\_postagger.tag(words)

#print (x)

return x

#return series.apply(lambda a: english\_postagger.tag(series))

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

pos = POS()

dp = DataManager('train.csv','test.csv')

#print (dp.trainTweets," waah2")

# train = pd.read\_csv(open('train.csv', 'rU'))

#test = pd.read\_csv(open('test.csv', 'rU'))

trainTargets = ["Hillary Clinton", "Legalization of Abortion", "Atheism", "Climate Change is a Real Concern", "Feminist Movement"]

testTargets = ["Hillary Clinton", "Legalization of Abortion", "Atheism", "Climate Change is a Real Concern",

"Feminist Movement", "Donald Trump"]

trainFile = ["hillary", "abortion", "atheism", "climate", "feminist"]

testFile = ["hillary", "abortion", "atheism", "climate", "feminist", "trump"]

f1 = []

for target in trainFile:

f1.append(open("train/"+target+"\_train.csv", "wb"))

for tweet in dp.trainTweets:

idx = trainTargets.index(tweet[1])

words = list(tweet[0])

stopSymbols = ["@", "#"]

for word in range(len(words)):

if words[word][0] in stopSymbols:

words[word] = words[word][1:]

#print ("f1 "+str(words) + "," + tweet[1] + "," + tweet[2])

f1[idx].write(str(words) + "," + tweet[1] + "," + tweet[2])

f1[idx].write("\n")

g1 = []

for target in testFile:

g1.append(open("test/"+target+"\_test.csv", "wb"))

for tweet in dp.testTweets:

idx = testTargets.index(tweet[1])

words = list(tweet[0])

stopSymbols = ["@", "#"]

for word in range(len(words)):

if words[word][0] in stopSymbols:

words[word] = words[word][1:]

#print ("g1 "+str(words) + "," + tweet[1] + "," + tweet[2])

g1[idx].write(str(words)+ "," + tweet[1] + "," + tweet[2])

g1[idx].write("\n")

#print train.Tweet

# train\_stfrd\_POStagged = []

# k = 0

# f = []

# for target in trainFile:

# f.append(open("train/"+target.lower()+"\_tagged\_train.txt", "wb"))

#

# g = []

# for target in testFile:

# g.append(open("test/"+target.lower()+"\_tagged\_test.txt", "wb"))

# for tweet in dp.trainTweets:

# idx = trainTargets.index(tweet[1])

# #f[idx] = open("train/"+tweet[1]+"\_pos\_tagged.txt")

# # getAns = pos\_tag(tweet)

# # test\_stfrd\_POStagged.append(getAns)

# x = pos\_tag(tweet)

# for row in x:

# #print row[0], row[1], tweet[1], tweet[2]

# f[idx].write(str(row[0]+","+row[1]+","+tweet[1]+","+tweet[2]))

# f[idx].write("\n")

# k += 1

#

# print(k)

# print("done train POS")

#

# k = 0

# for tweet in dp.testTweets:

# idx = testTargets.index(tweet[1])

# # f[idx] = open("train/"+tweet[1]+"\_pos\_tagged.txt")

# # getAns = pos\_tag(tweet)

# # test\_stfrd\_POStagged.append(getAns)

# x = pos\_tag(tweet)

# for row in x:

# # print row[0], row[1], tweet[1], tweet[2]

# g[idx].write(str(row[0] + "," + row[1] + "," + tweet[1] + "," + tweet[2]))

# g[idx].write("\n")

# k += 1

#

# print(k)

# print("done test POS")

# test\_stfrd\_POStagged = []

# k = 0

# f = open("test/pos\_tagged\_test.txt", "wb")

# for tweet in dp.testTweets:

# # getAns = pos\_tag(tweet)

# # test\_stfrd\_POStagged.append(getAns)

# x = pos\_tag(tweet)

# for row in x:

# #print row[0], row[1]

# f.write(str(row[0]+","+row[1]))

# f.write("\n")

# k += 1

# print(k)

#

# print("done test POS")

# train\_stfrd\_POStagged = pos\_tag(train.Tweet)

# print("done test POS")

#print (test\_stfrd\_POStagged)

makeDirs("train")

makeDirs("test")

makeDirs("3\_pos\_tags")

makeDirs("3\_pos\_tags/train")

makeDirs("3\_pos\_tags/test")

makeDirs("all\_words")

makeDirs("all\_words/train")

makeDirs("all\_words/test")

makeDirs("sublex\_all\_words")

makeDirs("sublex\_all\_words/train")

makeDirs("sublex\_all\_words/test")

# pos\_tagged\_train = open("train/pos\_tagged\_train.txt", 'wb')

# for tweet in dp.trainTweets:

# tweet = tweet[0]

# #words = tweet.split()

# # # relevant\_tokens = [w for w in words if '@' not in w and '#' not in w]

# # # #return (" ".join(relevant\_tokens))

# #

# # series = series.apply(lambda tweet: rem\_mentions\_hasht(tweet))

#

# # java\_path = "C:/Program Files/Java/jre1.8.0\_111/bin/java.exe"

# # os.environ['JAVAHOME'] = java\_path

# print(tweet)

# english\_postagger = CoreNLPPOSTagger('stanford-postagger-3.8.0.jar')

#

# #return tweet.apply(lambda a: english\_postagger.tag(nltk.word\_tokenize(a)))

# pos\_tagged\_train.write(tweet.apply(lambda a: english\_postagger.tag(CoreNLPTokenizer.tokenize(a))))

#

for file in trainFile:

pos.extract\_bow\_3\_pos\_tags("train/"+file+"\_tagged\_train.txt")

for file in testFile:

pos.extract\_bow\_3\_pos\_tags("test/"+file+"\_tagged\_test.txt")

# pos.extract\_bow\_3\_pos\_tags("train/feminist\_tagged\_train.txt")

# pos.extract\_bow\_3\_pos\_tags("test/feminist\_tagged\_test.txt")

# print (len(POS.nouns))

# print (len(POS.verbs))

# print (len(POS.adjs))

#print (len(POS.features\_3\_pos), "len")

for file in trainFile:

pos.create\_file("train/"+file+"\_train.csv",

"3\_pos\_tags/train/"+file+"\_train\_bow\_features.csv")

for file in testFile:

pos.create\_file("test/"+file+"\_test.csv",

"3\_pos\_tags/test/"+file+"\_test\_bow\_features.csv")

# pos.extract\_bow\_all\_words("train/hillary\_tagged\_train.txt")

# pos.extract\_bow\_all\_words("test/donald\_tagged\_test.txt")

#

# pos.create\_file("hillary\_train.csv",

# "all\_words/train/donald\_train\_bow\_all\_features.csv",

# all\_words=True)

# pos.create\_file("train\_test\_files/test/donald\_test.csv",

# "all\_words/test/donald\_test\_bow\_all\_features.csv",

# all\_words=True)

# pos.read\_subjectivity\_lexicons('subjectivity\_clues\_hltemnlp05/subjclueslen1-HLTEMNLP05.tff')

# pos.create\_features\_with\_sublex("feminist\_train.csv",

# "sublex\_all\_words/train/feminist\_train\_sublex\_all\_features.csv")

# pos.create\_features\_with\_sublex("train\_test\_files/test/feminist\_test.csv",

# "sublex\_all\_words/test/feminist\_test\_sublex\_all\_features.csv")

for file in trainFile:

print ("---"+file+"---")

pos.calculate\_baseline("3\_pos\_tags/train/" + file + "\_train\_bow\_features.csv",

"3\_pos\_tags/test/" + file + "\_test\_bow\_features.csv")

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

#pos.calculate\_baseline("train.csv", "test.csv")