Cathy Walsh CSC 665, Final Project 02 May 2017

```
In [123]: from os import path
          from glob import glob
          from bs4 import BeautifulSoup
          from nltk.tokenize import RegexpTokenizer
          from nltk import ngrams, word_tokenize
          import numpy as np
          import pandas as pd
          from scipy.spatial.distance import cosine, euclidean, hamming
          from scipy.sparse import csr_matrix, csc_matrix, linalg
          from sklearn.metrics import accuracy_score, precision_score, recall_scor
          e, f1_score
          from sklearn.feature_extraction.text import CountVectorizer
          from pickle import Pickler
          import pprint
```

Reading and Reformatting data

extracting entities from html, labeling entities, creating context ten grams

```
In [2]: def read_doc(fname):
            """
            Returns a text document as a string
            """
            f = open(fname, 'r')
            text = f.read()
            f.close()
            return text
```

```
In [3]: def get_entities(textf):
            """
            Uses beatiful soup to parse an html file that contains each entity s
        et in a table
            finds all of the 'table' itmems and pulls the table data containing
         the entities
            and appends each entity set to a list writes the contents of each en
        tity list to a
            file with the same topic name, with '_Table.html' removed and replac
        ed with '_entities.txt'
            """
            html = read_doc(textf)
            soup = BeautifulSoup(html, 'html.parser')
            entities = []
            for tab in list(soup.find_all('table')):
                td = tab.find_all('td')
                entities.append(td)
            efname = textf.strip('_Table.html')+'_entities.txt'
            o = open(efname, 'w')
            for i, entry in enumerate(entities):
                entity_list = []
                for item in list(entities[i]):
                    if not (str(item.contents[0])).isdigit():
                        entity_list.append((str(item.contents[0])))
                estr = '\t'.join(entity_list)
                o.write(estr +'\n\n')
            o.close()
```

In [4]:
```
def entity_triples(enfile):
    """
    reads an entity file and for each entity in a set of coreferent enti
ties,
    creates a triple with (entity, docID, entityID) where the docID is t
he first
    four characters of the name of the document, and the entity ID is th
e number
    corresponding to the order in which that group of entities appears i
n the file
    this way, the only entities that have the same IDs are the ones that
 corefer to
    one another.  for example if the first set of entities in document
 "Elephants.txt"
    is "elephant, he, the elephant" the triples will be: (elephant, Ele
p, 0),
    (he, Elep, 0), and (the elephant, Elep, 0)
    """
    fname = path.basename(enfile).strip('.txt') + "_triples.txt"
    o = open(fname, 'w')
    docID = path.basename(enfile)[:4]
    entID = 0

    entities = [es for es in read_doc(enfile).split('\n\n') if es != '']
    doc_ents = []
    for ent_set in entities:
        co_ents = [e for e in ent_set.split('\t')]
        for co in co_ents:
            etrp = tuple([co.lower(), docID, entID])
            doc_ents.append(etrp)
            o.write(co.lower() + '\t' + docID + '\t' + str(entID) +
'\n')
        entID+=1
        o.write('\n')
    o.close()
    return doc_ents
```

In [236]:
```
# data directories and text files
html_dir = "/Users/catherinewalsh/Desktop/Dropbox/DataScience/final_proj
ect/data/html/*"
ent_dir = "/Users/catherinewalsh/Desktop/Dropbox/DataScience/final_proje
ct/data/entities/elists/*"
tfile = "/Users/catherinewalsh/Desktop/Dropbox/DataScience/final_projec
t/data/text/all_text.txt"
ent_trp = "/Users/catherinewalsh/Desktop/Dropbox/DataScience/final_proje
ct/data/entities/entity_triples.txt"
tengramf = "/Users/catherinewalsh/Desktop/Dropbox/DataScience/final_proj
ect/data/text/tengram_contexts.txt"
gold_triples = "/Users/catherinewalsh/Desktop/Dropbox/DataScience/final_
project/data/entities/triples_gold/*"
output_triples = "/Users/catherinewalsh/Desktop/Dropbox/DataScience/fina
l_project/data/entities/predictions/raw/*"
eval_files = "/Users/catherinewalsh/Desktop/Dropbox/DataScience/final_pr
oject/data/entities/predictions/triples_guess/*"
```

In [6]:
```
# list of html files
html_files = glob(html_dir)

# all of the text files
all_text = read_doc(tfile)

# make the entity files for each text/html pair
for h in html_files:
    get_entities(h)
```

In [7]:
```
# make the list of entity triples, store these tuples in a file called
 "entity triples"
all_entities = []
for ef in glob(ent_dir):
    all_entities+=(entity_triples(ef))
f = open(ent_trp, 'w')
for e in all_entities:
    f.write(e[0] + '\t' + e[1] + '\t' + str(e[2]) + '\n')
    #n=1
f.close()
#all_entities
```

In [8]:
```
# make the context ten-grams from the text, and write each ten gram as a
 line to a file called "tengram_contexts.txt"
# there will be some "contexts" that don't actually occur in the text, b
ecause all of the text files were concatenated
tokenizer = RegexpTokenizer(r'\w+')
tokens = tokenizer.tokenize(all_text.lower())
tengrams = ngrams(tokens, 10)
tengrm = []
g = open(tengramf, 'w')
for grm in tengrams:
    tengram = ' '.join(list(grm))
    tengrm.append(tengram)
    g.write(tengram + '\n')
g.close()
```

Building the Co-occurrence Matrix

context-by-entities matrix, counts, SVD

In [9]:
```
entities = np.array(all_entities)
contexts = np.array(tengrm)
MMsp = np.ndarray(shape=(len(contexts), len(entities)))
MMsp.shape
```

Out[9]:
```
(53692, 6791)
```

```
In [293]: coref_array = pd.DataFrame.as_matrix(coref_sims)

          # load the matrix in batches of 10,000 rows
          rows0_10 = np.load("crf10.npy")
          rows10_20 = np.load("crf10_20.npy")
          rows20_30 = np.load("crf20_30.npy")
          rows30_40 = np.load("crf30_40.npy")
          rows40_end = np.load("crf40_end.npy")

          MMsp = np.vstack((rows0_10, rows10_20, rows20_30, rows30_40,
          rows40_end))
```

```
In [11]:  # change to row/column sparse matrix
          mmSp = csr_matrix.tocsr(csc_matrix.tocsc(MMsp))
          # get the SVD
          U, s, Vt = linalg.svds(mmSp, return_singular_vectors=True)
          svdM = U.dot(np.diag(s).dot(Vt))
          U.shape, s.shape, Vt.shape, svdM.shape

Out[11]:  ((53692, 6), (6,), (6, 6791), (53692, 6791))
```

Analysis

```
In [12]:  # Pandas data frame of SVD, with labeled columns and rows
          entity_labels = [str(triple) for triple in all_entities]
          coref_sims = pd.DataFrame(svdM, index=tengrm, columns=entity_labels)
```

```
In [295]: coref_sims[:5]
```

```
Out[295]:
```

	('glasgow', 'Aber', 0)	('glasgow', 'Aber', 0)	('the river forth', 'Aber', 1)	('the forth', 'Aber', 1)	('craigmore', 'Aber', 2)	('craigmore', 'Aber', 2)	('craig 'Aber'
aberfoyle is a village in the region of stirling scotland	0.000521	0.000521	0.000130	0.000317	0.001086	0.001086	0.001(
is a village in the region of stirling scotland northwest	0.000526	0.000526	0.000129	0.000318	0.001095	0.001095	0.001(
a village in the region of stirling scotland northwest of	0.000526	0.000526	0.000129	0.000318	0.001095	0.001095	0.001(
village in the region of stirling scotland northwest of glasgow	0.000526	0.000526	0.000129	0.000318	0.001095	0.001095	0.001(
in the region of stirling scotland northwest of glasgow the	0.000526	0.000526	0.000129	0.000318	0.001095	0.001095	0.001(

5 rows × 6791 columns

List of "Lead Entities"

In [15]:
```
# For each group of entities in all of the "entity triples" files, pick
 the first one to compare to all the other
# vectors. lead_dictionary = {doc1:[lead1, lead2, ..., leadn], doc2:[lea
d1, ..., leadn]..., docn:[]}
```

In [16]:
```
triples = glob(gold_triples)
def get_leaders(trpf):
    leaders = []
    tf = open(trpf, 'r')
    ent_groups = tf.read().split('\n\n')
    tf.close()
    for group in ent_groups:
        group = group.split('\n')
        leader = group[0].split('\t')
        leader_fmt = str(tuple([str(leader[0]), str(leader[1]), int(lead
er[2])]))
        leaders.append(leader_fmt)
    return leaders
```

In [17]:
```
def all_leaders(edirectory):
    leader_dict = {}
    for f in edirectory:
        fleads = get_leaders(f)
        docname = path.basename(f).strip('_entities_triples.txt')
        leader_dict.update({docname:fleads})
    return leader_dict
```

In [18]:
```
# 1408 lead entities
# 6791 columns
# 9,561,728 comparisons
lead_entities = all_leaders(triples)
leads = 0
for doc in lead_entities.keys():
    leads += len(lead_entities[doc])
```

In [19]:
```
# {docname:[[group1], [group2], [group3]]}
def doc_corefs(docname, leader_dict):
    document_corefs = []
    doc_ents = leader_dict[docname]
    for lead in doc_ents:
        corefs = get_similar(lead)
        document_corefs.append(corefs)
    doc_predictions = {docname:document_corefs}
    return doc_predictions
```

In [20]:
```
# coref_sims.columns.values[0]
def get_similar(entity_label):
    evect = coref_sims[entity_label]
    if len(evect.shape) > 1:
        evect = evect.iloc[:,0]
    similar = []
    for i in range(len(coref_sims.columns)):
        compare = coref_sims.iloc[:,i]
        if similarity(np.array(evect), np.array(compare)) >= 0.995 and c
oref_sims.columns.values[i] not in similar:
            similar.append(coref_sims.columns.values[i])
    return similar
```

In [21]:
```
def similarity(arr1, arr2):
    return 1 / (1 + euclidean(arr1, arr2))
```

In [81]:
```
docnames = sorted(lead_entities.keys())
```

Get the coref predictions for each document, write to file

predict1 = doc_corefs(docnames[0], lead_entities) a = open(str(docnames[0])+'.txt', 'w') a.write(str(predict1)) a.close() . . . predict30 = doc_corefs(docnames[29], lead_entities) gh = open(str(docnames[29])+'.txt', 'w') gh.write(str(predict30)) gh.close()

In [234]:
```
predictions = glob(output_triples)
#predictions
```

In [ ]:
```

```

Convert Predictions to Column Format

In [235]:
```
for file in predictions:
    f = open(file, 'r')
    txt = f.read()
    f.close()
    out = open(eval_files+path.basename(file).strip('.txt')+"_predict.tx
t", 'w')
    #print(path.basename(file).strip('.txt')+"_predict.txt")
    fmt = txt.split(':')[1].split('"], ["')
    all_items = []
    for item in fmt:
        item = item.split('", "')
        all_items.append(item)
    for lst in all_items:
        for i in lst:
            i = i.split(', ')
            out.write(i[0].strip("('") + '\t' + i[1].strip("''") + '\t'
+ i[2].strip(')') + '\n')
        out.write('\n')
    out.close()
```

Evaluate Predictions

In [291]:
```
"""
Look at each of the prediction files, if the predictions in a group matc
h the document and entity ID of the lead
entity, this is a corectly identified coreference, print: "match!"
"""
output = glob(eval_files)
total_correct = 0
for filepath in output:
    o = open(filepath, 'r')
    groups = o.read().split('\n\n')
    o.close()
    #print(path.basename(filepath).strip("_predict.txt"))
    for group in groups:
        guesses = group.split('\n')
        lead = guesses[0]
        if len(lead.split('\t'))==3:
            entID = lead.split('\t')[2]
            docID = lead.split('\t')[1]
            #print(lead)
            if len(guesses) > 1:
                for guess in guesses[1:]:
                    if len(guess.split('\t')) == 3:
                        guessENT = guess.split('\t')[2]
                        guessDOC = guess.split('\t')[1]
                        if guessID==guessENT and docID==guessDOC:
                            print(guess)
                            #print("match!")
                            #print('\n')
    #print('\n')
#print('\n')
```

In [ ]:

In [ ]: