Projet : Google Buzz: tf-idf, cosinus similarity, collocations

Table des matières

Introduction

1 Tests du code python contenu dans le chapitre

1.1 Extraction des données

```
import os
import sys
import buzz
from BeautifulSoup import BeautifulStoneSoup
from nltk import clean_html
import json
USER = "timoreilly";
MAX_RESULTS = 100
# Helper function for removing html and converting escaped entities.
# Returns UTF-8
def cleanHtml(html):
    return BeautifulStoneSoup(clean_html(html),

convertEntities=BeautifulStoneSoup.HTML_ENTITIES).contents[0]
client = buzz.Client()
posts_data = client.posts(type_id='@public',
user_id=USER,
max_results=MAX_RESULTS).data
posts = []
for p in posts_data:
    # Fetching lots of comments for lots of posts could take a little
    # bit of time. Thread pool code from mailboxes__CouchDBBulkReader.py could
    # be adapted for use here.
    comments = [{'name': c.actor.name, 'content': cleanHtml(c.content)} for c
    in p.comments().data]
    link = p.uri
    post = {
    'title': cleanHtml(p.title),
    'content': cleanHtml(p.content),
    'comments': comments,
    'link': link,
    }
    posts.append(post)
# Store out to a local file as json data if you prefer
if not os.path.isdir('out'):
    os.mkdir('out')
filename = os.path.join('out', USER + '.buzz')
f = open(filename, 'w')
f.write(json.dumps(posts))
f.close()
print >> sys.stderr, "Data written to", f.name
# Or store it somewhere like CouchDB like so...
# server = couchdb.Server('http://localhost:5984')
# DB = 'buzz-' + USER
# db = server.create(DB)
# db.update(posts, all_or_nothing=True)
```

Ce code permet de recupérer les données depuis google base en se servant d'une API (Buzz-Python-Client). Une fois qu'on les a récupérées, on les transforme en format Json pour les sauvegarder dans un fichier ou dans une base de données coucheDB.

Ces données sont spécialement constituées des tweets de certains internautes.

Le programme que l'on avait de base ne marchait pas. Car le lien fourni dans le chapitre n'existait plus, nous avons recherché sur internet pour récupérer un dataset pour la suite du projet.

1.2 Exploration rapide des données textuelles avec nltk

Le nltk est un package python pour le traitement automatique des langues naturelles. Nous avons tout d'abord effectué des commandes pour mieux nous y familiariser.

```
import xlrd
import nltk
from math import log

def loadData(filname):
    excel = xlrd.open_workbook(filname);
    sheet = excel.sheet_by_name("Sheet1");
    N = sheet.nrows;
    contenu = [];
    head = sheet.row_values(0);
    for i in range(1,N):
        line = sheet.row_values(i);
        dic ={};
        for j in range(len(line)):
            dic[head[j]]=line[j]
        contenu.append(dic)
    return contenu;

tweet_data = loadData("data.xlsx");

all_content = " ".join([p['Tweet content'] for p in tweet_data])
```

Nous avons testé 3 méthodes:

- concordance
- vocab
- collocations

Ces méthodes suffisent pour cette exploration rapide.

Pour la concordance, cela permet de lister les documents qui contiennent le mot passé en argument.

Pour le vocab, la fonction crée un objet qui hérite d'un dictionnaire où les clés sont les mots et les valeurs, leur fréquence.

Pour la collocation, on ne sait pas pour l'instant.

1.3 Tf - Idf

Tf signifie Term Frequency en anglais, c'est-à-dire la fréquence d'un mot dans un document (tweet).

Idf signifie Inverse Document Frequency en anglais, c'est-à-dire l'inverse de la fréquence d'un mot dans le corpus.

Tf - Idf est simplement le produit des 2.

```
# fonction qui calcule le TF
def tf(term, doc, normalize=True):
```

```
    doc = doc.lower().split()
    if normalize:
       return doc.count(term.lower()) / float(len(doc))
    else:
       return doc.count(term.lower()) / 1.0

# fonction qui calcule le IDF
def idf(term, corpus):
    num_texts_with_term = len([True for text in corpus if term.lower()
    in text.lower().split()])
    try:
        return 1.0 + log(float(len(corpus)) / num_texts_with_term)
    except ZeroDivisionError:
        return 1.0

 # fonction qui calcul le TF-IDF
 def tf_idf(term, doc, corpus):
    return tf(term, doc) * idf(term, corpus)
```

1.4 La similarité en cosinus

La similartié en cosinus est le fait de vérifier si 2 documents sont similaires.

```
all_posts = [post['Tweet content'].lower().split() for post in buzz_data]
# Provides tf/idf/tf_idf abstractions for scoring
tc = nltk.TextCollection(all_posts)
# Compute a term-document matrix such that td_matrix[doc_title][term]
# returns a tf-idf score for the term in the document
td_matrix = {}
for idx in range(len(all_posts)):
    post = all_posts[idx]
    fdist = nltk.FreqDist(post)
    doc_title = buzz_data[idx]['User Name']
    link = buzz_data[idx]['Tweet Id']
    td_matrix[(doc_title, link)] = {}
    for term in fdist.iterkeys():
        td_matrix[(doc_title, link)][term] = tc.tf_idf(term, post)
    # Build vectors such that term scores are in the same positions...

distances = {}
for (title1, link1) in td_matrix.keys():
    distances[(title1, link1)] = {}
    (max_score, most_similar) = (0.0, (None, None))
    for (title2, link2) in td_matrix.keys():
        # Take care not to mutate the original data structures
        # since we're in a loop and need the originals multiple times
        terms1 = td_matrix[(title1, link1)].copy()
        terms2 = td_matrix[(title2, link2)].copy()
        # Fill in "gaps" in each map so vectors of the same length can be
computed
    for term1 in terms1:
        if term1 not in terms2:
            terms2[term1] = 0
    for term2 in terms2:
        if term2 not in terms1:
            terms1[term2] = 0
```

```
    # Create vectors from term maps
    v1 = [score for (term, score) in sorted(terms1.items())]
    v2 = [score for (term, score) in sorted(terms2.items())]
    # Compute similarity among documents
    distances[(title1, link1)][(title2, link2)] = \
    nltk.cluster.util.cosine_distance(v1, v2)
    if link1 == link2:
        continue
    if distances[(title1, link1)][(title2, link2)] > max_score:
        (max_score, most_similar) = (distances[(title1,
link1)][(title2,link2)], (title2, link2))
    print( '''Most similar to %s (%s)\t%s (%s) \tscore %s ''' % (title1,
link1,most_similar[0], most_similar[1], max_score))
```

2 Améliorations apportées

Rien

3 Intérêts de l'étude

Le principal intérêt de notre étude est de savoir de façon rapide et précise les intérêts des utilisateurs afin de leur proposer des services adéquats et personnalisés. Nous vendrons donc ces programmes aux entreprises commerçantes qui auront plus facilement des clients.

Nous pouvons aussi proposer nos services à des entreprises qui cherchent à mettre des publicités personalisées sur de nombreux sites.

Conclusion