

Code

# PYTHON CODE:

import numpy as np

import pandas as pd

import time

import re

import requests

from bs4 import BeautifulSoup

searchword = 'financial'

start\_day = '20180101'

end\_day = '20181231'

filename = r'D:\kananovich\personal\OneDrive\Documents\family\jobsAnton\dataIncubator\stage02\myproject\data\scrapped\final\all.csv'

# The portions of the code below draw on the code originally written by nimolne

# (https://github.com/nilmolne/Text-Mining-The-New-York-Times-Articles),

# modified for my needs

def parse\_articles(articles):

news = []

for i in articles['response']['docs']:

dic = {}

dic['date'] = i['pub\_date'][0:10] # cutting time of day.

dic['atype'] = i['type\_of\_material']

dic['url'] = i['web\_url']

dic['word\_count'] = int(i['word\_count'])

dic['lead\_par'] = i['lead\_paragraph']

news.append(dic)

return news

def get\_articles\_url(api, start\_day, end\_day):

all\_articles = []

day = start\_day

print('Retrieving articles URL...'),

#Loop through all years of interest

while day != end\_day:

print(day)

# Some pages might return a 'No JSON object could be decoded'

# Example: country = Turkey, year = 1998, page 4

# To keep this error from stopping the loop a try/except was used.

for i in range(0,1):

try:

# Call API method with the parameters discussed on the README file

articles = api.search(q = {searchword},

fq = {'source':['The New York Times']},

begin\_date = day,

end\_date = day,

sort = 'oldest', page = str(i))

# Check if page is empty

if articles['response']['docs'] == []: break

articles = parse\_articles(articles)

all\_articles = all\_articles + articles

except Exception:

pass

# Avoid overwhelming the API

time.sleep(1)

day = get\_nextday(day)

# Copy all articles on the list to a Pandas dataframe

articles\_df = pd.DataFrame(all\_articles)

# Make sure we filter out non-news articles and remove 'atype' column

#articles\_df = articles\_df.drop(articles\_df[articles\_df.atype != 'News'].index)

#articles\_df.drop('atype', axis = 1, inplace = True)

# Discard non-working links (their number of word\_count is 0).

# Example: http://www.nytimes.com/2001/11/06/world/4-die-during-police-raid-in-istanbul.html

# articles\_df = articles\_df[articles\_df.word\_count != 0]

articles\_df = articles\_df.reset\_index(drop = True)

print('Done!')

return(articles\_df)

def scarp\_articles\_text(articles\_df):

# Unable false positive warning from Pandas dataframe manipulation

pd.options.mode.chained\_assignment = None

articles\_df['article\_text'] = 'NaN'

session = requests.Session()

print('Scarping articles body text...'),

for j in range(0, len(articles\_df)):

url = articles\_df['url'][j]

print('j=',j)

print(url)

req = session.get(url)

huyup = BeautifulSoup(req.text, 'lxml')

soup = huyup.find('section', attrs={'name':'articleBody'})

#initiate non-existing tag

#we will need it later to check if somothing was found

i\_dont\_exist = huyup.find('nonExistantTag')

notype = type(i\_dont\_exist)

# Get only HTLM tags with article content

# Articles through 1986 are found under different p tag

if type(soup) != notype:

paragraph\_tags = soup.find\_all('p')

if paragraph\_tags == []:

paragraph\_tags = huyup.find\_all('p', itemprop = 'articleBody')

# Put together all text from HTML p tags

article = ''

for p in paragraph\_tags:

article = article + ' ' + p.get\_text()

# Clean article replacing unicode characters

article = article.replace(u'\u2018', u"'").replace(u'\u2019', u"'").replace(u'\u201c', u'"').replace(u'\u201d', u'"')

#article = remCroco(article)

# Copy article's content to the dataframe

articles\_df['article\_text'][j] = article

print('Done!')

return articles\_df

# The code below is attributed to champerbarton

# (https://github.com/champebarton/NYTimesArticleAPI

# I had to use it explicitly instead of importing a module

# from nytimesarticle import articleAPI, because

# it hasn't worked from the imported module

API\_ROOT = 'http://api.nytimes.com/svc/search/v2/articlesearch.'

API\_SIGNUP\_PAGE = 'http://developer.nytimes.com/docs/reference/keys'

class NoAPIKeyException(Exception):

def \_\_init\_\_(self, value):

self.value = value

def \_\_str\_\_(self):

return repr(self.value)

class articleAPI(object):

def \_\_init\_\_(self, key = None):

"""

Initializes the articleAPI class with a developer key. Raises an exception if a key is not given.

Request a key at http://developer.nytimes.com/docs/reference/keys

:param key: New York Times Developer Key

"""

self.key = key

self.response\_format = 'json'

if self.key is None:

raise NoAPIKeyException('Warning: Missing API Key. Please visit ' + API\_SIGNUP\_PAGE + ' to register for a key.')

def \_bool\_encode(self, d):

"""

Converts boolean values to lowercase strings

"""

for k, v in d.items():

if isinstance(v, bool):

d[k] = str(v).lower()

return d

def \_options(self, \*\*kwargs):

"""

Formats search parameters/values for use with API

:param \\*\\*kwargs: search parameters/values

"""

def \_format\_fq(d):

for k,v in d.items():

if isinstance(v, list):

d[k] = ' '.join(map(lambda x: '"' + x + '"', v))

else:

d[k] = '"' + v + '"'

values = []

for k,v in d.items():

value = '%s:(%s)' % (k,v)

values.append(value)

values = ' AND '.join(values)

return values

kwargs = self.\_bool\_encode(kwargs)

values = ''

for k, v in kwargs.items():

if k is 'fq' and isinstance(v, dict):

v = \_format\_fq(v)

elif isinstance(v, list):

v = ','.join(v)

values += '%s=%s&' % (k, v)

return values

def search(self,

response\_format = None,

key = None,

\*\*kwargs):

"""

Calls the API and returns a dictionary of the search results

:param response\_format: the format that the API uses for its response,

includes JSON (.json) and JSONP (.jsonp).

Defaults to '.json'.

:param key: a developer key. Defaults to key given when the articleAPI class was initialized.

"""

if response\_format is None:

response\_format = self.response\_format

if key is None:

key = self.key

url = '%s%s?%sapi-key=%s' % (

API\_ROOT, response\_format, self.\_options(\*\*kwargs), key

)

self.req = requests.get(url)

return self.req.json()

def get\_nextday(today):

todayStruct = time.strptime(today, '%Y%m%d')

epochseconds = time.mktime(todayStruct)

deltaDay = 60\*60\*24

nextDayEpoch = epochseconds+deltaDay

nexDayStruct = time.localtime(nextDayEpoch)

nextDay= time.strftime('%Y%m%d', nexDayStruct)

return nextDay

api = articleAPI('UC0yZec7sBIqmDUSCvPnRLQYFso1bocD')

articles\_df = get\_articles\_url(api, start\_day, end\_day)

#articles\_df.tail()

articles\_df = scarp\_articles\_text(articles\_df)

#articles\_df.tail()

export\_csv = articles\_df.to\_csv (filename, index = None, header=True)

##############

# R CODE

install.packages("stm")

library(stm)

#set working directory

setwd("D:\kananovich\personal\OneDrive\Documents\family\jobsAnton\dataIncubator\stage02\myproject\data\scrapped\final\")

#load the data

lexisall <- read.csv("all.csv",header = TRUE)

#pre-process text

processed<-textProcessor(documents=lexisall$text,metadata=lexisall)

meta<-processed$meta

vocab<-processed$vocab

docs<-processed$documents

out <- prepDocuments(docs,vocab,meta)

#use prep documents.

docs<-out$documents

vocab<-out$vocab

meta <-out$meta

#set random seed

set.seed(02138)

#lda with k=10

mod10all <-

selectModel(

docs,

vocab,

K = 10,

data = meta,

max.em.its = 1000,

init.type = "LDA",

runs = 10

)

#lda with k=5

mod5all <-

selectModel(

docs,

vocab,

K = 5,

data = meta,

max.em.its = 1000,

init.type = "LDA",

runs = 5

)

##compare topic labels

#get topic labels

labelTopics(mod10all$runout[[1]], topics=NULL, n = 10, frexweight = 0.5)

labelTopics(mod10all$runout[[2]], topics=NULL, n = 10, frexweight = 0.5)

labelTopics(mod5all$runout[[1]], topics=NULL, n = 10, frexweight = 0.5)

labelTopics(mod5all$runout[[2]], topics=NULL, n = 10, frexweight = 0.5)

# mean numbers for semantic coherence

mean(mod10all$semcoh[[1]])

mean(mod10all$semcoh[[2]])

mean(mod5all$semcoh[[1]])

mean(mod5all$semcoh[[2]])

# mean numbers for exclusivity

mean(mod10all$exclusivity[[1]])

mean(mod10all$exclusivity[[2]])

mean(mod5all$exclusivity[[1]])

mean(mod5all$exclusivity[[2]])

##compare semantic coherence and exclusivity

## mod10all performed best

plotModels(mod10all)

plotModels(mod5all)

#save proportions of topics in each text in the corpus in a csv file

write.csv(mod10all$runout[[1]]$theta,"proportionsall.csv",row.names=FALSE)

**DESCRIPTION**

Stock markets have been a constant presence in media coverage. Their ups and downs supply journalists with drama and action, which are considered to be the key ingredients of a newsworthy story. This means that, although financial news is likely to be saturated with informational “noise,” it can also provide potentially useful insights into the stock markets’ mood. Not only that – and this is the key question that motivates my project – this opens up an opportunity that the news media can not only reflect but also predict the upcoming changes in stock-market performance.

Paraphrasing the popular metaphor that describes the news media as a watchdog, my project asks: Can this watchdog sense an upcoming stock-market swing and start barking before the swing arrives?

As a result of this project, I plan to develop a tool that can be used by investors to predict a change in stock prices based on the features of the current news coverage. Specifically, the tool will predict the closing values of S&P 500 – as a commonly followed stock-market index – based on the analysis of the coverage in The New York Times, as the national newspaper of record.

The project is to be implemented in two steps.

At STEP 1 – which I preliminarily executed – I tested the hypothesis that the NYT coverage does, indeed, reflect the S&P 500’s current performance.

To do that, I used the NYT API to retrieve the meta information of the articles published in the newspapers’ financial articles, and then scraped the full-text versions of the articles from the newspaper’s website. This returned 911 articles, which I saved as a .csv file.

I then ran a topic-modeling analysis of the articles in R. The analysis infered 10 topics, or themes with statistically distinct patters of occurrence of words, which I validated by reading the articles identified by the algorithm as most indicative of each topic, and summarized with a substantive label. Topics included domestic partisan politics (with indicatives stems, or word root forms, such as "democrat," "republican," "polit," "state," "trump," "elect," "parti," "campaign"), changes in foreign, mostly European, politics ("world," "new," "first," "time," "merkel," "macron," "european"), trade politics with China ("china," "trade," "percent," "tariff," "deficit," "govern"), technology ("facebook," "alibaba," "bitcoin," "tesla," "amazon," "spotifi"), etc.

Then, for each article, I identified the salience of each topic, as a proportion of text that exhibited the “traces” of the respective topic. After that, for each day in the timeframe, an average salience of each topic in the news coverage was calculated. Finally, to see which of the 10 topics did the best job matching the dynamic of the S&P values, I created 10 time-series plots that mapped the S&P 500’s closing values versus the daily salience of each topic in the news.

Figures 1 and 2 present two of them, which have provided the best match based on the preliminary visual analysis. Notably, both are related to political news, but appear to demonstrate different dynamics. An increase in foreign political coverage tended to coincide with the upward dynamic in S&P 500 prices (Figure 1). In contrast, the coverage of domestic news tended to behave countercyclically. With the notable exception on Nov. 6, the date of the midterm elections, when the journalists focused on partisan politics (Figure 2), this increase in coverage appears to have been accompanied by S&P 500’s poorer stock-market performance.

At STEP 2, to be implemented at The Data Incubator, I will subject this observation to more rigorous statistical testing, as well as will expand the sample beyond the financial news section. I will examine, specifically, which of the political topics that have not yet made their way to the financial news – where they would reflect concurrent price swings – have the biggest power in predicting the swings that’s yet to come.

I will use the results of this analysis to formalize the predictive model that will be used in the tool. Given the features of the news content that are updated in real-time and the planning horizon, the tool will predict the likelihood and directionality of a swing in the S&P values.

Links to data sets:

NYT articles: <http://developer.nytimes.com/apis>

S&P 500 prices: <http://finance.yahoo.com/quote/%5EGSPC/history/>