Appendix A – Code for Scraping PDF files

from bs4 import BeautifulSoup

from urllib2 import urlopen

import urllib

start\_year = 1997

stop\_year = 2015

root\_url = "http://elpub.architexturez.net/documents/series/ELPUB"

root\_site = "http://elpub.architexturez.net"

def download\_paper(url):

"""Searches for link to pdf and then download that pdf"""

html = urlopen(url).read()

soup = BeautifulSoup(html, "lxml")

links = soup.find\_all("div", class\_="pdfpreview-image-wrapper")

if (len(links) == 1):

if links[0].next\_element.has\_attr("href"):

print "downloading: " + links[0].next\_element["href"]

print "into file: " + links[0].next\_element["href"].split("f/")[1]

download\_file(links[0].next\_element["href"], links[0].next\_element["href"].split("f/")[1])

return 1

else:

return 0

elif (len(links) == 0):

return 0

else:

return 2

def get\_links\_to\_papers(url):

"""Finds links to each paper's page"""

html = urlopen(url).read()

soup = BeautifulSoup(html, "lxml")

paper\_links = soup.find\_all("span", class\_="biblio-title-chicago")

print "There are " + str(len(paper\_links)) + " papers in year " + str(year)

counter = 1

to\_return = []

for link in paper\_links:

if ((counter > 1) or (year < 2010)):

if link.next\_element.has\_attr('href'):

to\_return.append(link.next\_element['href'])

else:

print "ERROR, paper " + str(counter) + " in year " + str(year) + " seems to have no page!"

else:

print "In 2010 the links changed slightly"

counter += 1

return to\_return

def download\_file(url, filename):

urllib.urlretrieve(url, filename)

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

links = []

for year in range(start\_year, stop\_year+1):

current\_url = root\_url + str(year)

links = get\_links\_to\_papers(current\_url)

paper\_counter = 1

for x in links:

return\_code = download\_paper(root\_site + x)

if (return\_code == 0):

print "ERROR, no links found for paper " + str(counter) + " in year " + str(year)

elif (return\_code == 2):

print "ERROR, multiple links found for paper " + str(counter) + " in year " + str(year)

else:

print "success ", paper\_counter

paper\_counter += 1

Appendix B – Code for converting PDF files into Plain Text files

from subprocess import Popen, PIPE

from docx import opendocx, getdocumenttext

#http://stackoverflow.com/questions/5725278/python-help-using-pdfminer-as-a-library

from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter

from pdfminer.converter import TextConverter

from pdfminer.layout import LAParams

from pdfminer.pdfpage import PDFPage

from cStringIO import StringIO

def convert\_pdf\_to\_txt(path):

rsrcmgr = PDFResourceManager()

retstr = StringIO()

codec = 'utf-8'

laparams = LAParams()

device = TextConverter(rsrcmgr, retstr, codec = codec, laparams = laparams)

fp = file(path, 'rb')

interpreter = PDFPageInterpreter(rsrcmgr, device)

password = ""

maxpages = 0

caching = True

pagenos = set()

for page in PDFPage.get\_pages(fp, pagenos, maxpages = maxpages, password = password, caching = caching, check\_extractable = True):

interpreter.process\_page(page)

fp.close()

device.close()

str = retstr.getvalue()

retstr.close()

return str

def document\_to\_text(filename, file\_path):

if filename[-4:] == ".doc":

cmd = ['antiword', file\_path]

p = Popen(cmd, stdout=PIPE)

stdout, stderr = p.communicate()

return stdout.decode('ascii', 'ignore')

elif filename[-5:] == ".docx":

document = opendocx(file\_path)

paratextlist = getdocumenttext(document)

newparatextlist = []

for paratext in paratextlist:

newparatextlist.append(paratext.encode("utf-8"))

return '\n\n'.join(newparatextlist)

elif filename[-4:] == ".odt":

cmd = ['odt2txt', file\_path]

p = Popen(cmd, stdout=PIPE)

stdout, stderr = p.communicate()

return stdout.decode('ascii', 'ignore')

elif filename[-4:] == ".pdf":

return convert\_pdf\_to\_txt(file\_path)

Appendix C – Code for analyzing the data to give word frequencies over the different time periods

#load library

library(SnowballC)

library(tm)

library(ggplot2)

library(wordcloud)

#set working directory

setwd("C:/Users/ENTER\_DIRECTORY HERE")

#Create Corpus

docs <- Corpus(DirSource("C:/ENTER\_DIRECTORY\_HERE"))

#Get high level information of documents loaded in 'docs'

docs

#create the toSpace content transformer

toSpace <- content\_transformer(function(x, pattern) {return (gsub(pattern, " ", x))})

#clear specific characters by transforming them to spaces (getting rid of them only will join words between them)

docs <- tm\_map(docs, toSpace, "-")

docs <- tm\_map(docs, toSpace, ":")

docs <- tm\_map(docs, toSpace, "’")

docs <- tm\_map(docs, toSpace, "‘")

docs <- tm\_map(docs, toSpace, " -")

docs <- tm\_map(docs, toSpace, "%")

#Remove punctuation – replace punctuation marks with " "

docs <- tm\_map(docs, removePunctuation)

#Transform to lower case (need to wrap in content\_transformer)

docs <- tm\_map(docs,content\_transformer(tolower))

#Strip digits (std transformation, so no need for content\_transformer)

docs <- tm\_map(docs, removeNumbers)

#remove stopwords using the standard list in tm (the, and etc)

docs <- tm\_map(docs, removeWords, stopwords("english"))

#Strip whitespace (cosmetic?)

docs <- tm\_map(docs, stripWhitespace)

#Stem document

docs <- tm\_map(docs,stemDocument)

# as usual english and US spelling differences....

docs <- tm\_map(docs, content\_transformer(gsub), pattern = "visualis", replacement = "visual")

docs <- tm\_map(docs, content\_transformer(gsub), pattern = "visualiz", replacement = "visual")

docs <- tm\_map(docs, content\_transformer(gsub), pattern = "introduct", replacement = "introduc")

docs <- tm\_map(docs, content\_transformer(gsub), pattern = "analyst", replacement = "analyt")

docs <- tm\_map(docs, content\_transformer(gsub), pattern = "scientist", replacement = "scienc")

docs <- tm\_map(docs, content\_transformer(gsub), pattern = "creativ", replacement = "creat")

#Creating a document term matrix

dtm <- DocumentTermMatrix(docs)

#check out the dtm

#dtm

# inspect a particular document

# writeLines(as.character(docs[[1]]))

#inspect particular documents and terms

#inspect(dtm[1:2,1000:1005])

# getting the frequency of occurance of each word

freq <- colSums(as.matrix(dtm))

#Sort freq in descending order of term count.

ord <- order(freq,decreasing = TRUE)

# limit to terms occuring between x and y documents and restrict word length to between 2 and 20 characters - change the c(,) to get which occur in most

dtmr <-DocumentTermMatrix(docs, control=list(wordLengths=c(2, 20), bounds = list(global = c(10,59))))

#inspect(dtmr)

#cumulative frequencies of words across documents and sort

freqr <- colSums(as.matrix(dtmr))

ord <- order(freqr,decreasing=TRUE)

#chart

#change font size of axis

theme\_set(theme\_grey(base\_size = 25))

wf=data.frame(term=names(freq),occurrences=freq)

p <- ggplot(subset(wf, freq>25), aes(term, occurrences))

p <- p + geom\_bar(stat="identity")

p <- p + theme(axis.text.x=element\_text(angle=45, hjust=1))

#wordcloud

set.seed(42)

#wordcloud(names(freq),freq, min.freq = 1)

#inspect(dtmr)