Introduction to Web Science: Assignment #4

Dr. Nelson

Alexander Nwala

Friday, May 1, 2015

Contents

Problem 1

Using the pages from A3 that boilerpipe successfully processed, download those representations again & reprocess them with boilerpipe.
Document the time difference (e.g., Time(A4) Time(A3)). Compute the Jaccard Distance x for each pair of pages (i.e., P(A3) & P(A4) for:

1. Unique terms (i.e., unigrams)
2. Bigrams
3. Trigrams

See: http://en.wikipedia.org/wiki/Jaccard_index. For each of the 3 cases (i.e., 1-, 2-, 3-grams) build a Cumulative Distribution Function that shows the % change on the x-axis & the % of the population on the x-axis. See: http://en.wikipedia.org/wiki/Cumulative_distribution_function. Give 3-4 examples illustrating the range of change that you have measured.

SOLUTION 1

The solution for this problem is outlined by the following steps:

1. **Download pages a second time:** The first set of text (**linksFile.txt**) was downloaded on March 29, 2015 and the second was downloaded on April 10, 2015 (12 days apart).

2. **Tokenize and calculate n-grams:** Due to Listing 2. the text downloaded in 1. was to tokenized and 1, 2, 3-grams calculated for both sets (text downloaded on March 29, 2015 and April 10, 2015).

3. **Calculated Jaccard distance:** Also Due to Listing 2. the Jaccard distance was calculated for every pair in the set across the 1, 2, 3-grams.

4. **Plot Cummulative Distribution Function For Jaccard Distance:** Chart 1. was produced due to Listing 1.

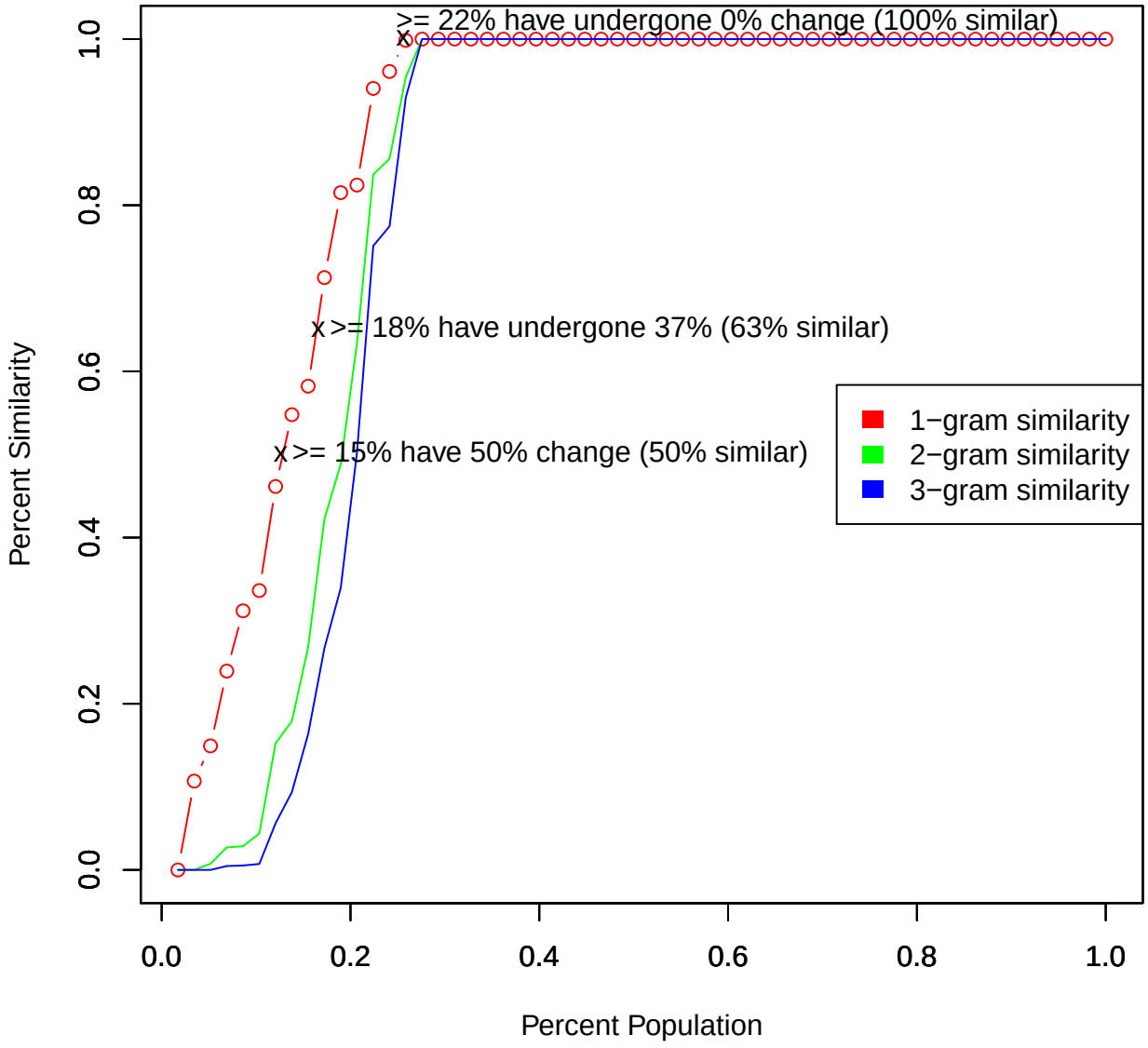
Listing 1: Plot CDF for Jaccard Distance values

```
#!/usr/bin/env Rscript

similarityValues <- read.table('nGramSimilarity.txt', header=T)

# Create the data.
a <- similarityValues$oneGramSim
b <- similarityValues$twoGramSim
c <- similarityValues$threeGramSim

# Set colors for the CDF.
aCDFcolor <- rgb(1,0,0)
bCDFcolor <- rgb(0,1,0)
cCDFcolor <- rgb(0,0,1)
```

Problem 1 continued on next page...

```
# Create a single chart with all 3 CDF plots.
#plot(ecdf(a), col=aCDFcolor, main=NA)
#plot(ecdf(b), col=bCDFcolor, add=T)
#plot(ecdf(c), col=cCDFcolor, add=T)
n = sum(!is.na(similarityValues$oneGramSim))

oneGramSim = sort(similarityValues$oneGramSim)
twoGramSim = sort(similarityValues$twoGramSim)
threeGramSim = sort(similarityValues$threeGramSim)

plot((1:n)/n, oneGramSim, type = 'b', ylim = c(0, 1), xlab = 'Percent Population',
    ylab = 'Percent Similarity', main = 'Chart 1: Empirical Cummulative Distribution\n1
    -, 2-, 3- grams similarity between page pairs at time points: \nMarch 29, 2015 and
    April 10, 2015', col=aCDFcolor)
par(new=TRUE)
plot((1:n)/n, twoGramSim, type = 'l', ylim = c(0, 1), xlab = '', ylab = '', main = '',
     col=bCDFcolor)
par(new=TRUE)
plot((1:n)/n, threeGramSim, type = 'l', ylim = c(0, 1), xlab = '', ylab = '', main = '
    ', col=cCDFcolor)

text(0.23, 1.02, '>= 22% have undergone 0% change (100% similar)', pos=4, col='black')
text(0.23, 1, 'x', pos=4, col='black')

text(0.16, 0.65, '>= 18% have undergone 37% (63% similar)', pos=4, col='black')
text(0.14, 0.65, 'x', pos=4, col='black')

text(0.12, 0.5, '>= 15% have 50% change (50% similar)', pos=4, col='black')
text(0.1, 0.5, 'x', pos=4, col='black')

legend('right', c('1-gram similarity', '2-gram similarity', '3-gram similarity'), fill
    =c(aCDFcolor, bCDFcolor, cCDFcolor), border=NA)

#For each of the 3 cases (i.e., 1-, 2-, 3-grams) build a Cumulative Distribution
    Function that shows the % change on the x-axis & the % of the population on the x-
    axis
```

Chart 1: Empirical Cummulative Distribution
1−, 2−, 3− grams similarity between page pairs at time points:
March 29, 2015 and April 10, 2015



Listing 2: Calculate Jaccard Distance

```
# precondition: all tuples are from a set - no repetition
def computeJaccardSimilarity(firstListOfTuples, secondListOfTuples):

     similarity = -1
     if( len(firstListOfTuples) != 0 and len(secondListOfTuples) != 0 ):

          intersectionCount = 0
          unionDict = {}

          for firstTuple in firstListOfTuples:
               for secondTuple in secondListOfTuples:
                    if( len(firstTuple) == len(secondTuple) ):

                         unionDict.setdefault(firstTuple, 0)
                         unionDict.setdefault(secondTuple, 0)

                         if( firstTuple == secondTuple ):
                              intersectionCount = intersectionCount + 1
          similarity = intersectionCount / (float) (len(unionDict))

     return similarity

def genericSimilarity(s0Dict, s1Dict):

     #print computeJaccardSimilarity( flt0, flt1 )
     if( len(s0Dict) != 0 and len(s1Dict) != 0 ):

          count = 0
          for url, plaintext in s0Dict.items():
               if url in s1Dict:
                    count += 1
                    allStrings = consolidateListOfStringsAndMakeSet( s0Dict[url] )
                    s0Tokens = allStrings.split(' ')
                    s0Tokens = list(set(s0Tokens))

                    allStrings = consolidateListOfStringsAndMakeSet( s1Dict[url] )
                    s1Tokens = allStrings.split(' ')
                    s1Tokens = list(set(s1Tokens))

                    firstVersionOneGrams = find_ngrams(s0Tokens, 1)
                    firstVersionTwoGrams = find_ngrams(s0Tokens, 2)
                    firstVersionThreeGrams = find_ngrams(s0Tokens, 3)

                    secondVersionOneGrams = find_ngrams(s1Tokens, 1)
                    secondVersionTwoGrams = find_ngrams(s1Tokens, 2)
                    secondVersionThreeGrams = find_ngrams(s1Tokens, 3)

                    oneGramSimilarity = computeJaccardSimilarity(firstVersionOneGrams,
                         secondVersionOneGrams)

                    twoGramSimilarity = computeJaccardSimilarity(firstVersionTwoGrams,
                         secondVersionTwoGrams)
```

Problem 1 continued on next page...

```
                threeGramSimilarity = computeJaccardSimilarity(
                    firstVersionThreeGrams, secondVersionThreeGrams)

                print count, (oneGramSimilarity*100), (twoGramSimilarity*100), (
                    threeGramSimilarity*100)
            else:
                print 'NOT: ', url
```

The file nGramSimilarity.txt contains the similarity values

Problem 2

Using the pages from Q1 (A4), download all TimeMaps (including TimeMaps with 404 responses, i.e. empty or null TimeMaps). Upload all the TimeMaps to github Build a CDF for # of mementos for each original URI (i.e., x-axis = # of mementos, y-axis = % of links) See: http://timetravel.mementoweb.org/guide/api/

SOLUTION 2

The solution for this problem is outlined by the following steps:

1. **Download all timemaps:** Due to Listing 4, the timemaps for all URLs was downloaded based on the memento API as implemented by https://github.com/anwala/wdill. For each URI, Listing 4. paginates through all the timemaps and dereferences each in order to count the mementos.

2. **Plot CDF for # of mementos:** Listing 3. plots the CDF for all the count of mementos by using R's primitive Empirical Cummulative Distribution function (ECDF). As seen from Chart 2, the distribution tells a story or two extremes: most of the URIs have 0 mementos due to the fact that they are new, and have had insufficient chances to be archived. On the otherhand, a few of the URIs are very popular, hence had sufficient time to be archived.

The folder Timemaps contains the timemaps

Listing 3: Plot CDF for Memento Count

```
#!/usr/bin/env Rscript

mementoCount <- read.table('mementoCount.txt', header=T)

# Create the data.
a <- mementoCount$MementoCount

# Set colors for the CDF.
aCDFcolor <- rgb(1,0,0)
n = sum(!is.na(mementoCount$MementoCount))
```

Problem 2 continued on next page...

```
#sortedMementoCount = sort(mementoCount$MementoCount)
#plot(sortedMementoCount, type = 'l', ylim = c(0, 1), xlab = '# of mementos', ylab =
    '% of links', main = 'Empirical Cummulative Distribution Mementos', col=aCDFcolor)

plot(ecdf(a), col=aCDFcolor, xlab='# of mementos', ylab='% of links', main='Chart 2:
    Empirical Cummulative Distribution Mementos')
```

Listing 4: Count Mementos for all URIs

```
def getMementosPages(url):

     pages = []
     url = url.strip()
     if(len(url)>0):

          firstChoiceAggregator = "http://timetravel.mementoweb.org/timemap/json/"
          timemapPrefix = firstChoiceAggregator + url
          #timemapPrefix = 'http://mementoproxy.cs.odu.edu/aggr/timemap/link/1/' + url

          '''
               The CS memento aggregator payload format:
                    [memento, ..., memento, timemap1]; timemap1 points to next page
               The LANL memento aggregator payload format:
                    1. [timemap1, ..., timemapN]; timemapX points to mementos list
                    2. [memento1, ..., mementoN]; for small payloads

               For LANL Aggregator: The reason the link format is used after
                   retrieving the payload
                                    with json format is due to the fact that the
                                        underlying code is based
                                    on the link format structure. json format was
                                         not always the norm
          '''



          #select an aggregator - start
          aggregatorSelector = ''

          co = 'curl --silent -I ' + timemapPrefix
          head = commands.getoutput(co)

          indexOfFirstNewLine = head.find('\n')
          if( indexOfFirstNewLine > -1 ):

               if( head[:indexOfFirstNewLine].split(' ')[1] != '200' ):
                    firstChoiceAggregator = "http://mementoproxy.cs.odu.edu/aggr/
                        timemap/link/1/"
                    timemapPrefix = firstChoiceAggregator + url

          if( firstChoiceAggregator.find('cs.odu.edu') > -1 ):
               aggregatorSelector = 'CS'
```

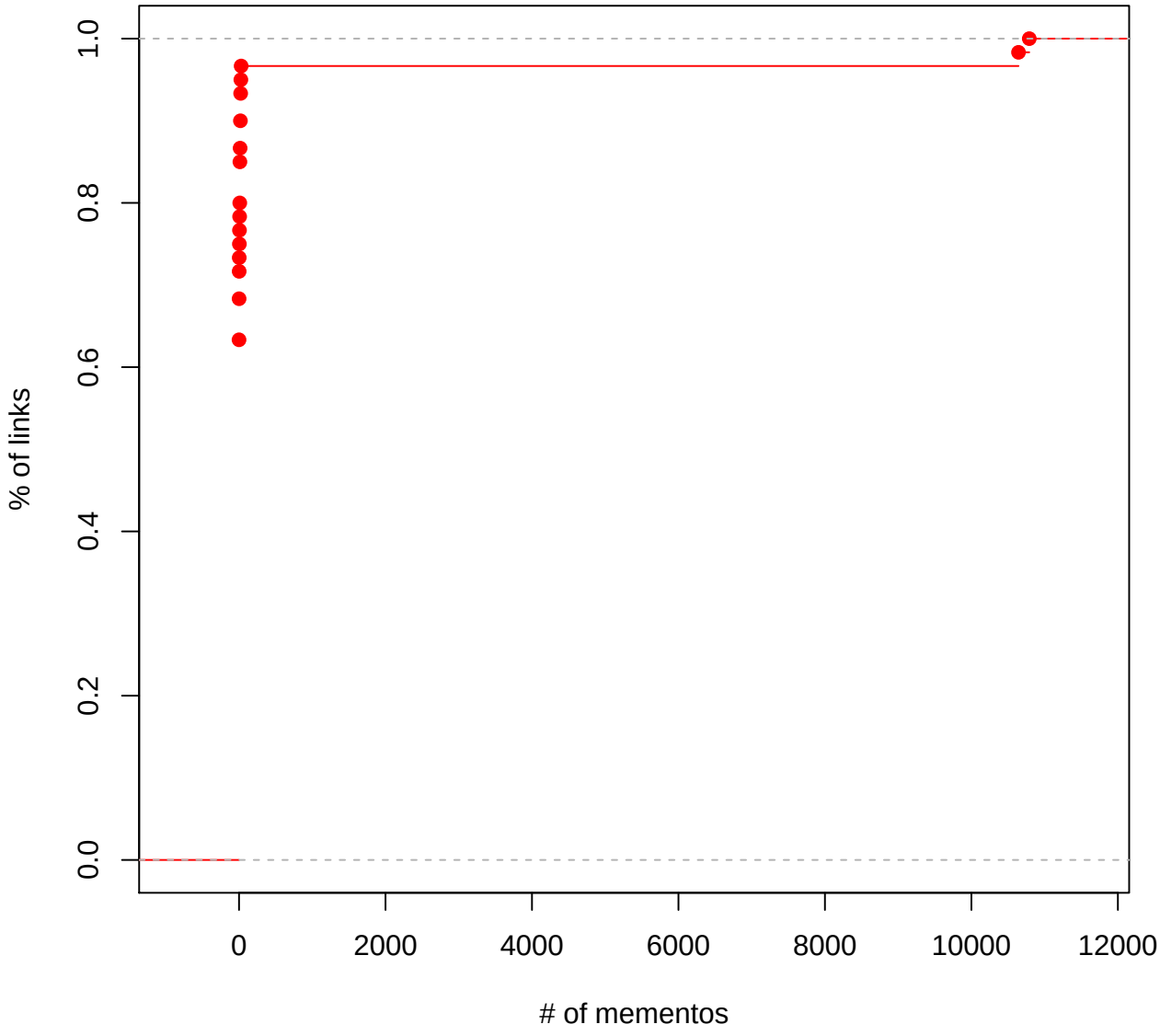
Problem 2 continued on next page...

```
        else:
            aggregatorSelector = 'LANL'

        #print '...using aggregator:', aggregatorSelector
        #select an aggregator - end

        #CS aggregator
        if( aggregatorSelector == 'CS' ):
            while( True ):
                #old: co = 'curl --silent ' + timemapPrefix
                #old: page = commands.getoutput(co)


                page = ''
                r = requests.get(timemapPrefix)
                print 'status code:', r.status_code
                if( r.status_code == 200 ):
                    page = r.text

                pages.append(page)
                indexOfRelTimemapMarker = page.rfind('>;rel="timemap"')

                if( indexOfRelTimemapMarker == -1 ):
                    break
                else:
                    #retrieve next timemap for next page of mementos e.g retrieve
                        url from <http://mementoproxy.cs.odu.edu/aggr/timemap/
                       link/10001/http://www.cnn.com>;rel="timemap"
                    i = indexOfRelTimemapMarker -1
                    timemapPrefix = ''
                    while( i > -1 ):
                        if(page[i] != '<'):
                            timemapPrefix = page[i] + timemapPrefix
                        else:
                            break
                        i = i - 1
        else:
            #LANL Aggregator
            #old: co = 'curl --silent ' + timemapPrefix
            #old: page = commands.getoutput(co)

            page = ''
            r = requests.get(timemapPrefix)
            if( r.status_code == 200 ):
                page = r.text

            try:
                payload = json.loads(page)

                if 'timemap_index' in payload:

                    for timemap in payload['timemap_index']:

```

```
                                timemapLink = timemap['uri'].replace('/timemap/json/', '
                                    /timemap/link/')
                                #old: co = 'curl --silent ' + timemapLink
                                #old: page = commands.getoutput(co)
                                #old: pages.append(page)
                                r = requests.get(timemapLink)
                                if( r.status_code == 200 ):
                                     pages.append(r.text)

                           elif 'mementos' in payload:
                                #untested block
                                timemapLink = payload['timemap_uri']['json_format'].replace('
                                    /timemap/json/', '/timemap/link/')
                                #old: co = 'curl --silent ' + timemapLink
                                #old: page = commands.getoutput(co)
                                #old: pages.append(page)

                                print 'timemap:', timemapLink
                                r = requests.get(timemapLink)
                                if( r.status_code == 200 ):
                                     pages.append(r.text)

                     except:
                          exc_type, exc_obj, exc_tb = sys.exc_info()
                          fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
                          print(fname, exc_tb.tb_lineno, sys.exc_info() )

          return pages

def getItemGivenSignature(page):

          listOfItems = []
          listOfItemsDateTime = []
          if( len(page) > 0 ):
               page = page.splitlines()
               for line in page:
                    if(line.find('memento";') != -1):
                         #uriRelDateTime: ['<http://www.webcitation.org/64ta04WpM>', ' rel
                             ="first memento"', ' datetime="Mon, 23 Jan 2012 02:01:29 GMT
                             ",']
                         uriRelDateTime = line.split(';')
                         if( len(uriRelDateTime) > 2 ):
                              if( uriRelDateTime[0].find('://') != -1 ):
                                   if( uriRelDateTime[2].find('datetime="') != -1 ):


                                        uri = ''
                                        uri = uriRelDateTime[0].split('<')
                                        #print uri
                                        if( len(uri) > 1 ):
                                             uri = uri[1].replace('>', '')
                                             uri = uri.strip()
```

```
                                        datetime = ''
                                        datetime = uriRelDateTime[2].split('"')
                                        if( len(datetime) > 1 ):
                                             datetime = datetime[1]

                                        if( len(uri) != 0 and len(datetime) != 0 ):
                                             #print uri, '---', datetime
                                             #listOfItems.append(uri +
                                                 globalMementoUrlDateTimeDelimeter +
                                                 datetime)
                                             listOfItems.append(uri)
                                             listOfItemsDateTime.append(datetime)

     return listOfItems, listOfItemsDateTime

def countMementosForUrl(url):

     url = url.strip()
     if( len(url) == 0 ):
          return 0

     count = 0
     #print "...getting memento pages"
     pages = getMementosPages(url)
     #print "...done getting memento pages"
     if(len(pages) != 0):
          #print 'pages:', len(pages)
          for i in range(0,len(pages)):
               mementos = getItemGivenSignature(pages[i])
               count += len(mementos)

     return count
```

Chart 2: Empirical Cummulative Distribution Mementos



```
The file mementoCount.txt contains the count of all
mementos for all the URIs in linksFile.txt.
```

Problem 3

Using 20 links that have TimeMaps With >= 20 mementos, have existed >= 2 years (i.e., Memento-Datetime of "first memento" is April XX, 2013 or older). Note: select from Q1/Q2 links, else choose them by hand.

For each link, create a graph that shows Jaccard Distance, relative to the first memento, through time x-axis: continuous time, y-axis: Jaccard Distance relative to the first memento

SOLUTION 3

The solution for this problem is outlined by the following steps:

1. **Download first memento:** This was achieved due to Listing 5. which downloads all the mementos for a given URI and sorts the entries in ascending order based on datetime values of the mementos.

   Consider the following: based on the initial list of URIs, only 21 had mementos, but only 4 (10790, 25, 29, 22) had memento count exceeding 20. From this short list, boilerplate removal was only successful for two (http://tinyurl.com/ and http://www-01.ibm.com/software/analytics/solutions/customer-analytics/social-media-analytics/). The solution to part 3 of this assignment addressed the two URIs. However, the solution could scale to address a larger set.

Listing 5: Get First Memento

```
def getFirstMemento(url):

    url = url.strip()
    if( len(url) == 0 ):
        return ('','')

    dictionaryOfMementos = {}
    count = 0
    #print "...getting memento pages"
    #pages is timemaps
    pages = getMementosPages(url)
    #print "...done getting memento pages"
    if(len(pages) != 0):
        #print 'pages:', len(pages)
        for i in range(0,len(pages)):
            mementos = getItemGivenSignature(pages[i])
            for m in mementos:
                mementoDatetime = m.split(globalMementoUrlDateTimeDelimeter);

                memento = mementoDatetime[0].strip()
                datetimeValue = str(mementoDatetime[1].strip())
                datetimeValue = datetime.strptime(datetimeValue, '%a, %d %b %Y
                    %H:%M:%S %Z')
```

```
                    dictionaryOfMementos[memento] = datetimeValue

    keys = sorted(dictionaryOfMementos, key=dictionaryOfMementos.get)

    if( len(keys) != 0 ):
        return (keys[0], dictionaryOfMementos[keys[0]])
    else:
        return ('','')
```

2. **Download text of first memento URI:** Due to Listing 6. based on justText [1] boilerpipe, the text for the first memento was downloaded.

Listing 6: Extract Text for URI of first Memento

```
def extractTextForURI(URI):
    page = urllib2.urlopen(URI).read()
    paragraphs = justext.justext(page, justext.get_stoplist('English'))

    allText = ''
    for paragraph in paragraphs:
        if paragraph['class'] == 'good':
            processedText = paragraph['text']
            processedText = processedText.encode('ascii', 'ignore')

            allText += processedText

    return allText
```

3. **Tokenize and compute similarity:** Due to Listing 7. which is similar to Listing 2. the 1-gram similarity was calculate between all mementos relative to the first memento for the selected URIs.

Listing 7: Calculate Jaccard Distance Relative to first Mementos

```
url1 = 'http://tinyurl.com/'
url1FirstMemento = 'http://web.archive.org/web/20020212130833/http://tinyurl.com/'

outputFile = open('sim1.txt', 'a')

allStringsFirstMemento = extractTextForURI(url1FirstMemento)
tokens = allStringsFirstMemento.split(' ')
tokens = list(set(tokens))
firstMementoVersionOneGrams = find_ngrams(tokens, 1)

pages = getMementosPages(url1)
if(len(pages) != 0):

    for i in range(0,len(pages)):
        mementos, listOfItemsDateTime = getItemGivenSignature(pages[i])

        for j in range(0, len(mementos)):
            url = mementos[j]
            if( url1FirstMemento != url ):

                #print url
```

```
                    allStrings = extractTextForURI(url)
                    s0Tokens = allStrings.split(' ')
                    s0Tokens = list(set(s0Tokens))
                    subsequentVersionOneGrams = find_ngrams(s0Tokens, 1)

                    #print firstMementoVersionOneGrams
                    #print
                    #print subsequentVersionOneGrams

                    oneGramSimilarity = computeJaccardSimilarity(
                        firstMementoVersionOneGrams, subsequentVersionOneGrams)
                    print i, len(pages), oneGramSimilarity, listOfItemsDateTime[j]
                    outputFile.write(str(oneGramSimilarity) + ' <> ' +
                        listOfItemsDateTime[j] + '\n')
```

4. **Plot Jaccard Distance:** Due to Listing 8. Chart's 3 and 4 were produced.

Listing 8: Plot Jaccard Distance Relative to first Mementos

```
#!/usr/bin/env Rscript

similarityValues <- read.table('sim1Plot.txt', header=T)

# Create the data.
a <- similarityValues$Sim1

# Set colors for the CDF.
aCDFcolor <- rgb(1,0,0)

plot(a, type = 'l', ylim = c(0, 1), xlab = 'Dates', ylab = 'Similarity', main = '\
    nChart 3: Jaccard Distance Relative to first Memento for
     tinyurl.com from \nMon, 13 Oct 2008 to Wed, 29 Apr 2015\n', col=aCDFcolor)


#For each of the 3 cases (i.e., 1-, 2-, 3-grams) build a Cumulative Distribution
    Function that shows the % change on the x-axis & the % of the population on the x-
    axis
```

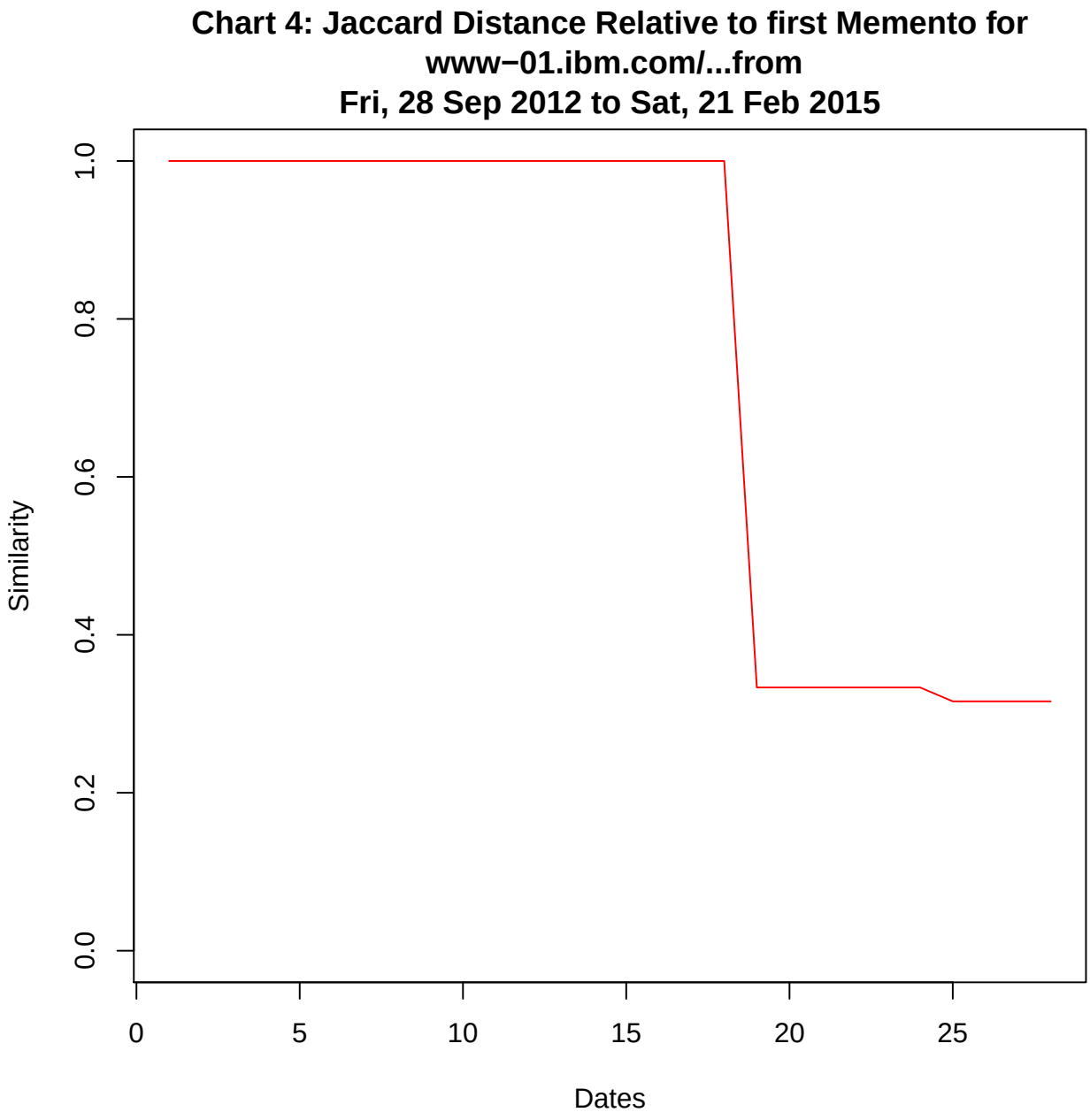


Chart 4: Jaccard Distance Relative to first Memento for
www−01.ibm.com/...from
Fri, 28 Sep 2012 to Sat, 21 Feb 2015
Similarity
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
Dates

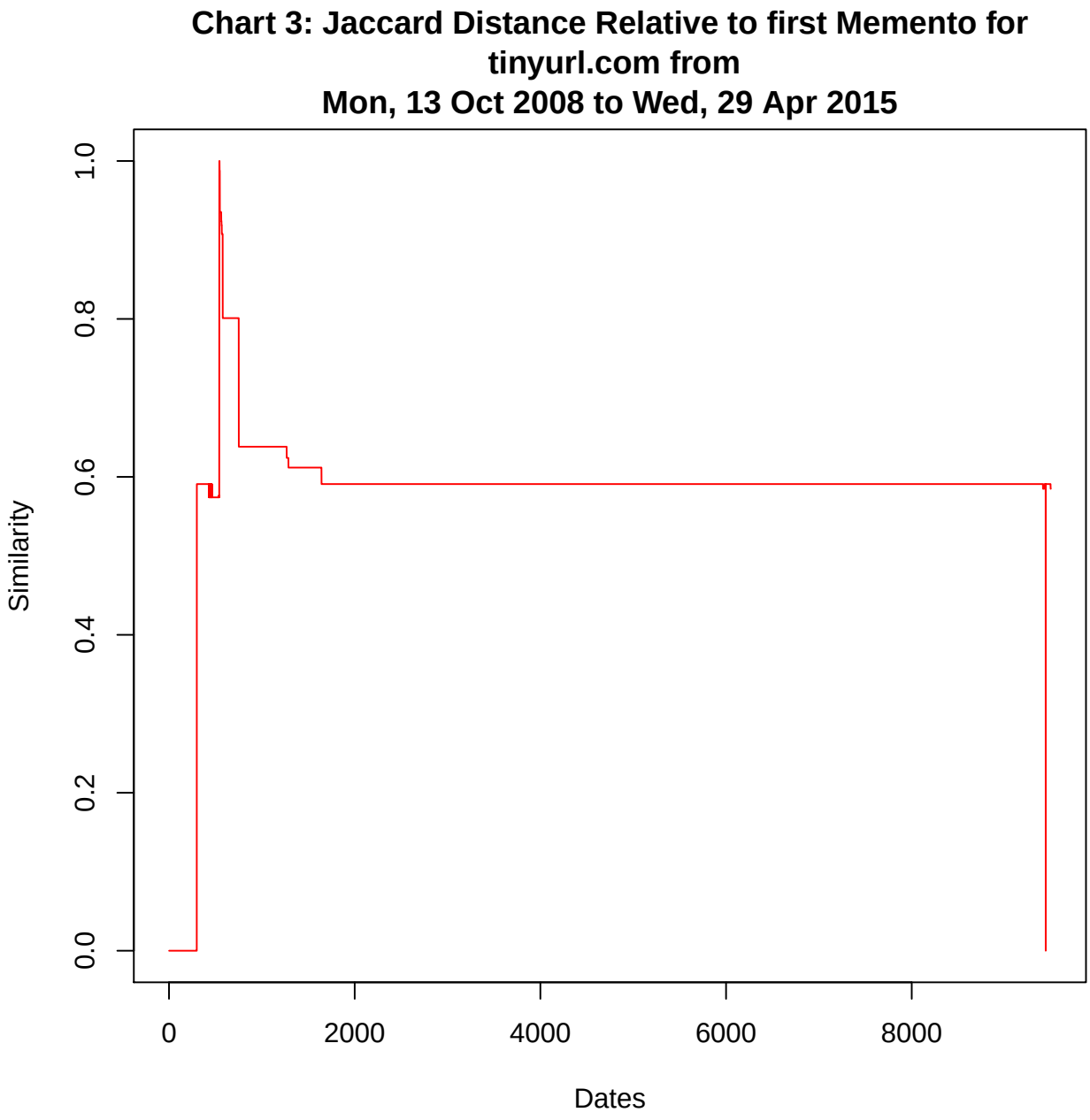


Chart 3: Jaccard Distance Relative to first Memento for
tinyurl.com from
Mon, 13 Oct 2008 to Wed, 29 Apr 2015
Similarity
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
Dates

```
The files sim1.txt and sim2.txt contains the similarity values
```

References

[1] jusText. https://pypi.python.org/pypi/jusText/2.0.0. Accessed: 2015-04-01.