CS 595: Assignment #9

Due on Thursday, December 4, 2014

Dr. Nelson 4:20pm

Holly Harkins

Contents

Problem 1

Create a blog-term matrix. Start by grabbing 100 blogs; include:

http://f-measure.blogspot.com/
http://ws-dl.blogspot.com/

and grab 98 more as per the method shown in class.

Use the blog title as the identifier for each blog (and row of the matrix). Use the terms from every item/title (RSS) or entry/title (Atom) for the columns of the matrix. The values are the frequency of occurrence. Essentially you are replicating the format of the "blogdata.txt" file included with the PCI book code. Limit the number of terms to the most "popular" (i.e., frequent) 500 terms, this is *after* the criteria on p. 32 (slide 7) has been satisfied.

Create a histogram of how many pages each blog has (e.g., 30 blogs with just one page, 27 with two pages, 29 with 3 pages and so on).

Answer:

I used generateURLs.py to capture the 98 Blogs (feedlist.txt).

Next step was to create a matrix. The matrix required was using blog title as the identifier for each blog row and using the terms for the columns of the matrix. The values are the frequency of occurrence. Using generatefeedvector.py, TFIDF is calculated as the score that balances the frequency of a term in a document vs. frequency of a term in all the documents. A limit to the number of terms to the most "popular" 500 terms (blogdata.txt).

To create the Histogram, I used getNextPage.py to generate a count of how many pages each blog has (numberOfPages.txt). Unfortunately I ran out of time and did not generate the count. There was an error in the code I could not resolve. I would have used R code to generate the histogram.

Listing 1: Generating Blog URLs

```
#!/usr/bin/python3
import sys
from bs4 import BeautifulSoup
import urllib.request
from urllib.parse import urlparse
```

Problem 1 continued on next page...

```
DEFAULT_COUNT=98
DEFAULT_SEED_URL='http://www.blogger.com/next-blog?navBar=true&blogID
    =3471633091411211117'
if len(sys.argv) != 3:
    print('Pass the blog count, defaulting to ' + str(DEFAULT_COUNT))
    print('Pass the seed URL, defaulting to ' + DEFAULT_SEED_URL)
    count=DEFAULT_COUNT
    url=DEFAULT_SEED_URL
else:
    count=sys.argv[1]
    url=sys.argv[2]

def parse(link):
    response = urllib.request.urlopen(link)
    soup = BeautifulSoup(response.read())
    response.close()
    return soup

def addNext(url,s):
    try:
        soup=parse(url)
        for atom in soup.findAll('link',rel='alternate',type='application/atom+xml'):
            atomHref=str(atom['href']).strip()
            s.add(atomHref)
            print('Added atom href: ' + atomHref)
    except:
        print('Exception parsing URL, skipping to next')
        pass

s=set()
while len(s) < count:
    addNext(url,s)
with open('feedlist.txt', 'w') as f:
    for atom in s:
        f.write(atom + '\n')
```

Listing 2: Generating Matrix

```
#!/usr/bin/python3
import feedparser
import re

# Returns title and dictionary of word counts for an RSS feed
def getwordcounts(url):
    # Parse the feed
    d=feedparser.parse(url)
    wc={}

    # Loop over all the entries
    for e in d.entries:
        if 'summary' in e: summary=e.summary
        else: summary=e.description

        # Extract a list of words
```

```
        words=getwords(e.title+' '+summary)
        for word in words:
            wc.setdefault(word,0)
            wc[word]+=1
    return d.feed.title,wc

def getwords(html):
    # Remove all the HTML tags
    txt=re.compile(r'<[^>]+>').sub('',html)

    # Split words by all non-alpha characters
    words=re.compile(r'[^A-Z^a-z]+').split(txt)

    # Convert to lowercase
    return [word.lower() for word in words if word!='']


apcount={}
wordcounts={}
feedlist=[line for line in file('feedlist.txt')]
for feedurl in feedlist:
    try:
        title,wc=getwordcounts(feedurl)
        wordcounts[title]=wc
        for word,count in wc.items():
            apcount.setdefault(word,0)
            if count>1:
                apcount[word]+=1
    except:
        print 'Failed to parse feed %s' % feedurl

#modification on the TFIDF calculations
wordlist=[]
for w,bc in sorted(apcount.items(), key=lambda t: t[1], reverse=True):
    frac=float(bc)/len(feedlist)
    if frac>0.1 and frac<0.5:
        wordlist.append(w)

out=file('blogdata.txt','w')
out.write('Blog')

#Limit the number of terms to the most "popular" 500 terms
wordlist=wordlist[0:500]
for word in wordlist: out.write('\t%s' % word)
out.write('\n')
for blog,wc in wordcounts.items():
    print blog
    out.write(blog)
    for word in wordlist:
        if word in wc: out.write('\t%d' % wc[word])
        else: out.write('\t0')
    out.write('\n')
print 'Total number of words=' +str(len(wordlist))
```

Problem 2

Create an ASCII and JPEG dendrogram that clusters (i.e., HAC) the most similar blogs (see slides 12 & 13). Include the JPEG in your report and upload the ascii file to github (it will be too unwieldy for inclusion in the report).

Answer:

Assignment9.py uses Pearsons correlation and averages the data for the 2 old clusters. The final cluster returned can be searched recursively to recreate all the clusters and their end nodes (ASCII.txt).

The drawdendrogram function uses the Python Imaging Library to generate Clust_Dendrogram.jpg.

Below is the dendogram that was created from the blog matrix.

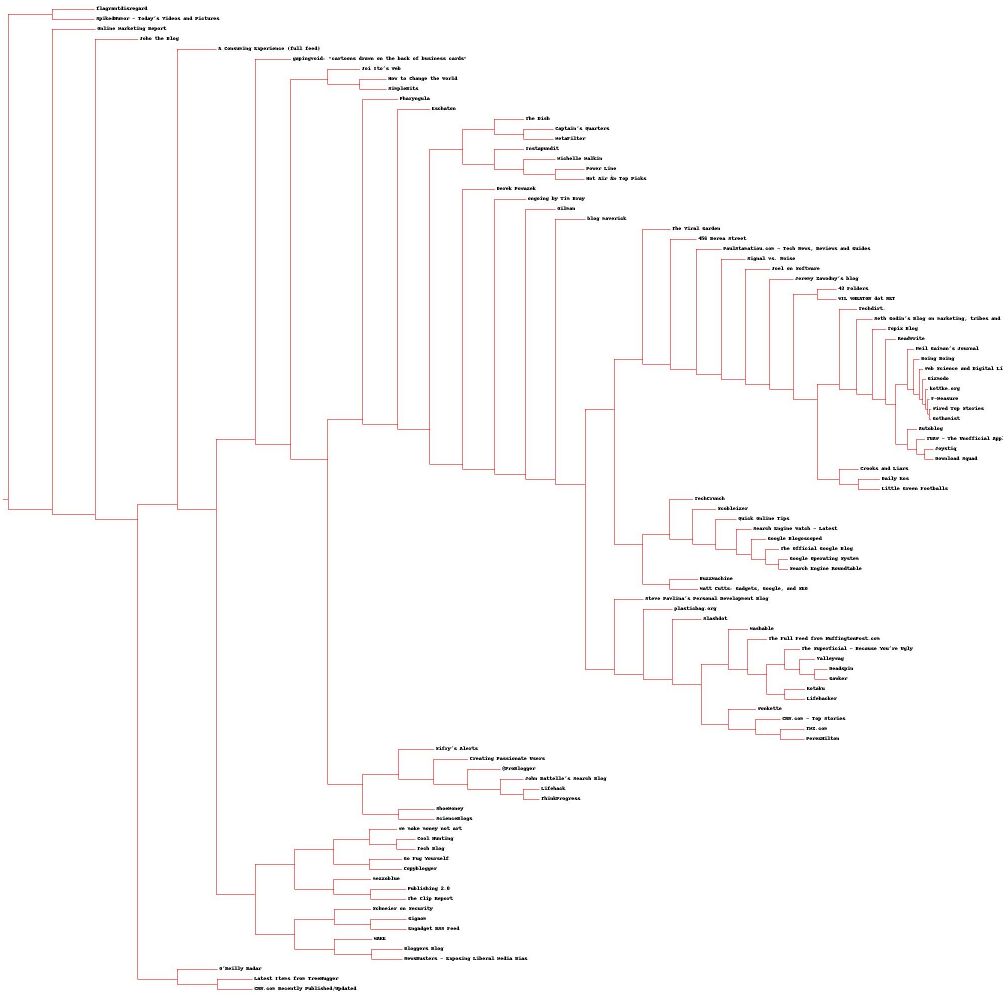

Figure 1: Clust Dendrogram

Problem 3

Cluster the blogs using K-Means, using k=5,10,20. (see slide 18).
How many interations were required for each value of k?

Answer:
Assignment9.py uses kcluster function to place a K points randomly in space that represent the centre of the cluster and assigns each blog to the nearest one. Then moves centroids to the average location of all the nodes that assigned to them. Repeats this process until the assignments stop changing.

For K = 5, there were 5 iterations needed to cluster the blogs. For k = 10, there were 6 iterations needed to cluster the blogs. For k=20, there were 5 iterations needed to cluster the blogs. The output generated is found in Kinterations.txt.

Problem 4

Use MDS to create a JPEG of the blogs similar to slide 29.
How many iterations were required?

Answer:
Assignment9.py uses 2 functions to create the MDS. Scaledown function takes the data vector obtained from readfile function, finds the difference between each pair of blogs, and matches them to a distances using Pearson correlation. These distances will represent the distances between the blogs in the chart. Returns the X and Y coordinates of the blogs on the two-dimensional chart.

Function draw2d uses Python Imaging Library to to generate an image with all the labels of all the different blogs plotted at the coordinates of that blog which obtained from first function.

Problem 4 continued on next page...

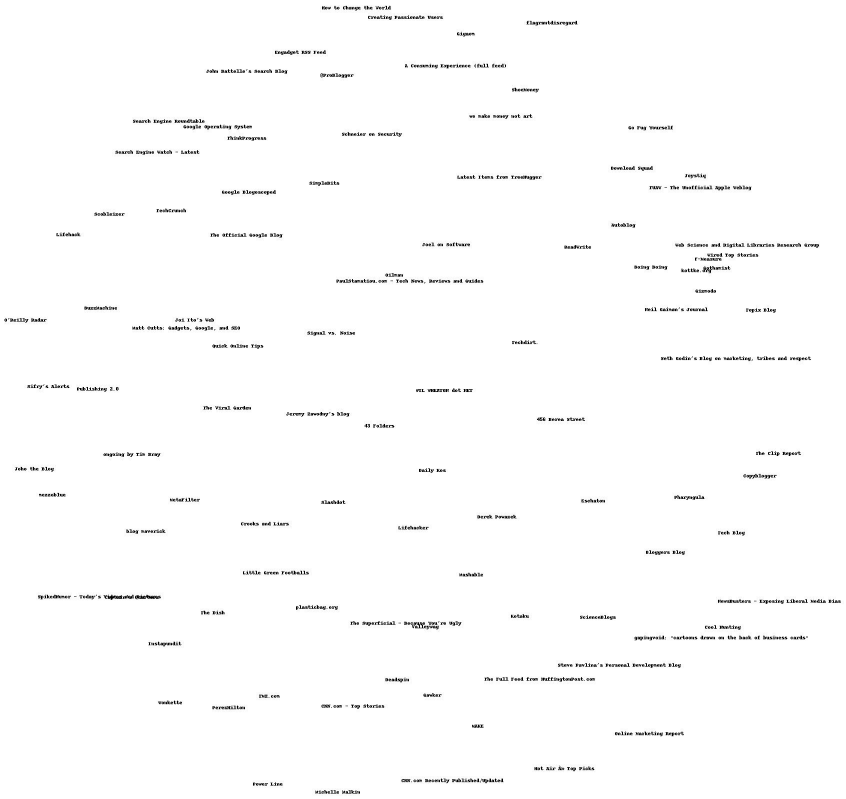

Figure 2: MDS

Listing 3: Assignment9 Python

```
import sys
import random
from PIL import Image,ImageDraw
from math import sqrt

f = open('ASCII.txt','w')

def readfile(filename):
     lines=[line for line in file(filename)]
     # First line is the column titles
     colnames=lines[0].strip().split('\t')[1:]
     rownames=[]
     data=[]
     for line in lines[1:]:
          p=line.strip().split('\t')
          # First column in each row is the rowname
          rownames.append(p[0])
```

```
            # The data for this row is the remainder of the row
            data.append([float(x) for x in p[1:]])
    return rownames,colnames,data


def pearson(v1,v2):
# Simple sums
    sum1=sum(v1)
    sum2=sum(v2)
# Sums of the squares
    sum1Sq=sum([pow(v,2) for v in v1])
    sum2Sq=sum([pow(v,2) for v in v2])
# Sum of the products
    pSum=sum([v1[i]*v2[i] for i in range(len(v1))])
# Calculate r (Pearson score)
    num=pSum-(sum1*sum2/len(v1))
    den=sqrt((sum1Sq-pow(sum1,2)/len(v1))*(sum2Sq-pow(sum2,2)/len(v1)))
    if den==0: return 0
    return 1.0-num/den

class bicluster:
    def __init__(self,vec,left=None,right=None,distance=0.0,id=None):
        self.left=left
        self.right=right
        self.vec=vec
        self.id=id
        self.distance=distance

def hcluster(rows,distance=pearson):

    distances={}
    currentclustid=-1
# Clusters are initially just the rows
    clust=[bicluster(rows[i],id=i) for i in range(len(rows))]
    while len(clust)>1:
        lowestpair=(0,1)
        closest=distance(clust[0].vec,clust[1].vec)
    # loop through every pair looking for the smallest distance
        for i in range(len(clust)):
            for j in range(i+1,len(clust)):
    # distances is the cache of distance calculations
                if (clust[i].id,clust[j].id) not in distances:
                    distances[(clust[i].id,clust[j].id)]= distance(clust[i].vec,
                        clust[j].vec)
                d=distances[(clust[i].id,clust[j].id)]
                if d<closest:
                    closest=d
                    lowestpair=(i,j)
    # calculate the average of the two clusters
        mergevec=[(clust[lowestpair[0]].vec[i]+ clust[lowestpair[1]].vec[i]) /2.0
            for i in range(len(clust[0].vec))]
    # create the new cluster
        newcluster=bicluster(mergevec,left=clust[lowestpair[0]],
```

Problem 4 continued on next page...

```
            right=clust[lowestpair[1]],
            distance=closest,id=currentclustid)
        # cluster ids that weren't in the original set are negative
            currentclustid-=1
            del clust[lowestpair[1]]
            del clust[lowestpair[0]]
            clust.append(newcluster)
        return clust[0]
def printclust(clust,labels=None,n=0):
        # indent to make a hierarchy layout

        for i in range(n):  f.write (' '),
        if clust.id<0:
        # negative id means that this is branch
            f.write ('-\n')
        else:
        # positive id means that this is an endpoint
            if labels==None:
                f.write (clust.id+'\n')
            else:
                f.write (labels[clust.id]+'\n')
        # now print the right and left branches
        if clust.left!=None:     printclust(clust.left,labels=labels,n=n+1)
        if clust.right!=None:    printclust(clust.right,labels=labels,n=n+1)


def getheight(clust):
        # Is this an endpoint? Then the height is just 1
        if clust.left==None and clust.right==None:
            return 1
        # Otherwise the height is the same of the heights of
        # each branch
        return getheight(clust.left)+getheight(clust.right)

def getdepth(clust):
        # The distance of an endpoint is 0.0
        if clust.left==None and clust.right==None:
            return 0
        # The distance of a branch is the greater of its two sides
        # plus its own distance
        return max(getdepth(clust.left),getdepth(clust.right))+clust.distance

def drawdendrogram(clust,labels,jpeg='clusters.jpg'):
        # height and width
        h=getheight(clust)*20
        w=2000
        depth=getdepth(clust)
        # width is fixed, so scale distances accordingly
        scaling=float(w-150)/depth
        # Create a new image with a white background
        img=Image.new('RGB',(w,h),(255,255,255))
        draw=ImageDraw.Draw(img)
        draw.line((0,h/2,10,h/2),fill=(255,0,0))
```

```
     # Draw the first node
     drawnode(draw,clust,10,(h/2),scaling,labels)
     img.save(jpeg,'JPEG')

def drawnode(draw,clust,x,y,scaling,labels):
     if clust.id<0:
          h1=getheight(clust.left)*20
          h2=getheight(clust.right)*20
          top=y-(h1+h2)/2
          bottom=y+(h1+h2)/2

     # Line length
          ll=clust.distance*scaling
     # Vertical line from this cluster to children
          draw.line((x,top+h1/2,x,bottom-h2/2),fill=(255,0,0))
     # Horizontal line to left item
          draw.line((x,top+h1/2,x+ll,top+h1/2),fill=(255,0,0))
     # Horizontal line to right item
          draw.line((x,bottom-h2/2,x+ll,bottom-h2/2),fill=(255,0,0))
     # Call the function to draw the left and right nodes
          drawnode(draw,clust.left,x+ll,top+h1/2,scaling,labels)
          drawnode(draw,clust.right,x+ll,bottom-h2/2,scaling,labels)
     else:
     # If this is an endpoint, draw the item label
          draw.text((x+5,y-7),labels[clust.id],(0,0,0))


def kcluster(rows,distance=pearson,k=4):
     # Determine the minimum and maximum values for each point
     ranges=[(min([row[i] for row in rows]),max([row[i] for row in rows]))
     for i in range(len(rows[0]))]
     # Create k randomly placed centroids
     clusters=[[random.random()*(ranges[i][1]-ranges[i][0])+ranges[i][0]
     for i in range(len(rows[0]))] for j in range(k)]
     lastmatches=None
     for t in range(100):
          print 'Iteration %d' % t
          bestmatches=[[] for i in range(k)]
     # Find which centroid is the closest for each row
          for j in range(len(rows)):
               row=rows[j]
               bestmatch=0
               for i in range(k):
                    d=distance(clusters[i],row)
                    if d<distance(clusters[bestmatch],row): bestmatch=i
               bestmatches[bestmatch].append(j)

     # If the results are the same as last time, this is complete
          if bestmatches==lastmatches: break
          lastmatches=bestmatches
     # Move the centroids to the average of their members
          for i in range(k):
               avgs=[0.0]*len(rows[0])
```

Problem 4 continued on next page...

```
                if len(bestmatches[i])>0:
                    for rowid in bestmatches[i]:
                        for m in range(len(rows[rowid])):
                            avgs[m]+=rows[rowid][m]
                    for j in range(len(avgs)):
                        avgs[j]/=len(bestmatches[i])
                    clusters[i]=avgs
    return bestmatches

def scaledown(data,distance=pearson,rate=0.01):
    n=len(data)
    # The real distances between every pair of items
    realdist=[[distance(data[i],data[j]) for j in range(n)] for i in range(0,n)]
    # Randomly initialize the starting points of the locations in 2D
    loc=[[random.random(),random.random()] for i in range(n)]
    fakedist=[[0.0 for j in range(n)] for i in range(n)]
    lasterror=None
    for m in range(0,1000):
    # Find projected distances
        for i in range(n):
            for j in range(n):
                fakedist[i][j]=sqrt(sum([pow(loc[i][x]-loc[j][x],2)
                    for x in range(len(loc[i]))]))
    # Move points
        grad=[[0.0,0.0] for i in range(n)]

        totalerror=0
        for k in range(n):
            for j in range(n):
                if j==k: continue
    # The error is percent difference between the distances
                errorterm=(fakedist[j][k]-realdist[j][k])/ realdist[j][k]
    # Each point needs to be moved away from or towards the other
    # point in proportion to how much error it has
                grad[k][0]+=((loc[k][0]-loc[j][0])/ fakedist[j][k])*errorterm
                grad[k][1]+=((loc[k][1]-loc[j][1])/ fakedist[j][k])*errorterm
    # Keep track of the total error
                totalerror+=abs(errorterm)
        print totalerror
    # If the answer got worse by moving the points, we are done
        if lasterror and lasterror<totalerror: break
        lasterror=totalerror
    # Move each of the points by the learning rate times the gradient
        for k in range(n):
            loc[k][0]-=rate*grad[k][0]
            loc[k][1]-=rate*grad[k][1]
    return loc

def draw2d(data,labels,jpeg='mds2d.jpg'):
    img=Image.new('RGB',(2000,2000),(255,255,255))
    draw=ImageDraw.Draw(img)
    for i in range(len(data)):
        x=(data[i][0]+0.5)*1000
```

```
            y=(data[i][1]+0.5)*1000
            draw.text((x,y),labels[i],(0,0,0))
    img.save(jpeg,'JPEG')

def main():
# Start Q2 Creating an ASCII and JPEG dendrogram
    blognames,words,data=readfile('blogdata.txt')
    cluster=hcluster(data)

    printclust(cluster,labels=blognames)
    f.close()
    drawdendrogram(cluster,blognames,jpeg='Clust_Dendrogram.jpg')


# Start Q3 using K-Means Clustering
    print "Clustering the blogs using k=5"
    kclust=kcluster(data,k=5)
    print "\nClustering the blogs using k=10"
    kclust=kcluster(data,k=10)
    print "\nClustering the blogs using k=20"
    kclust=kcluster(data,k=20)


# Start Q4 MDS
    print "\nMDS Creation"
    coords=scaledown(data)
    draw2d(coords,blognames,jpeg='MDS.jpg')


main();
```

References

[1] https://github.com/nico/collectiveintelligence-book/blob/master/feedlist.txt.