CS595 Assignment 10

Jon Robison

December 11, 2013

Q1.
Choose a blog or a newsfeed (or something similar with an Atom or RSS feed). It should be on a topic or topics of which you are qualified to provide classification training data. Find something with at least 100 entries Create between four and eight different categories for the entries in the feed:..download and process.

The program getTitles.py retrieves the paginated results and outputs to a pipe separated text file. This file is then manually processed to include necessary training information. This is accomplished by putting the feature in the 3rd location and the actual category in the 4th location (after the third pipe) when appropriate.

See program given in Appendix A.

Q2.
Manually classify the first 50 entries, and then classify (using the fisher classifier) the remaining 50 entries. Create a table with the title, the string used for classification, cprob(), predicted category, and actual category.

The driver program invokes docclass, reading the file given from A. This trains the classifier with the first half of whatever is given in the file, and guesses the last half. Different results are printed to a latex file and the console for the results table and debugging information/status, respectively.

Title	Feature	Predicted	Category	CP
ALBUM: FALL CITY FALL - VICTUS	FALL CITY	album	album	0.0
ALBUM: HANDBOOK - TITANOMACHY	HANDBOOK	album	album	0.0
ALBUM : WIHT - THE HARROWING OF THE NORTH	WIHT	album	album	0.0
SINGLE: IVORY SEAS - STILL BROODING / MOTHERS TONGUE	IVORY SEAS	single	single	0.0
EXCLUSIVE: ALCOPOP! Cat in a Hat 2013 Compilation #30	Cat in	event	event	0.0
SINGLE: MARIKA HACKMAN - BATH IS BLACK	MARIKA HACKMAN	single	single	0.0
FUTURE SOUNDS OF 2013	FUTURE SOUNDS	event	event	0.0
THE BLOG SOUND OF 2013 - THE TOP 5	BLOG SOUND	event	event	0.0
MUSIC LIBERATION TOP 10 ALBUMS OF 2012	TOP 10	event	event	0.0
Merry Christmas 2012 : First Aid Kit - Blue Christmas	Merry Christmas	event	event	0.0
MUSIC LIBERATION - TOP 10 EPS OF 2012	EP'S OF	event	event	0.0
MUSIC LIBERATIONS SINGLE OF THE YEAR 2012	OF THE YEAR	event	event	0.0
THE BLOG SOUND OF 2013	THE BLOG	event	event	0.0
NEW MUSIC: MARIKA HACKMAN	MARIKA HACKMAN	new	new	0.0
EP: NINETAILS - SLEPT AND DID NOT SLEEP	NINETAILS	album	album	0.0
EP: TO KILL A KING - WORD OF MOUTH	KILL A KING	album	album	0.0
EP: OLILOQUY - THE RED EP	THE RED	album	album	0.0
ALBUM: CITIZENS - CTZNS	CITIZENS	album	album	0.0
ALBUM: SUP PEEPS - START COLLECTING	SUP PEEPS	album	album	0.0
ALBUM: TALL SHIPS - EVERYTHING TOUCHING	EVERYTHING TOUCHING	album	album	0.0
ALBUM: NEGATIVE PEGASUS - LOOMING	NEGATIVE PEGASUS	album	album	0.0
SINGLE: LAURA WELSH - CALL TO ARMS / HOLLOW DRUM	LAURA WELSH	single	single	0.0
EP: CARA MITCHELL - HAVE YOU EVER WONDERED	CARA MITCHELL	album	album	0.0
SINGLE REVIEW: DEAD WOLF CLUB - RADAR	DEAD WOLF CLUB	album	single	0.0
LIVE REVIEW : Lucy Rose at the Jazz Cafe (Camden, London, 4th February 2012)	Lucy Rose at	album	event	0.0
Music Liberation and ooShirts.com	and ooShirts.com	interview	event	0.0
ALBUM REVIEW: CLOUD NOTHINGS - ATTACK ON MEMORY	CLOUD NOTHINGS	album	album	0.0
New Music : The Cast Of Cheers	Cast Of Cheers	interview	new	0.0
Future Sounds of 2012	Future	event	event	0.0
Music Liberations Writers takeover week : Moker	Writers takeover	album	event	0.0
Music Liberations Writers takeover week : Clive Rozario	Clive Rozario	album	event	0.0
Music Liberations Writers takeover week : Chris Meredith	Chris Meredith	album	event	0.0
Music Liberations Writers takeover week : Tom Nash	Tom Nash	album	event	0.0
The Blog Sound of 2012 : The Top 5	Top 5	album	event	0.0
Music Liberations Top 10 Albums of 2011	Albums of	album	event	0.0
Music Liberations Top 5 EPs of 2011	EPs of 2011	album	event	0.0
Music Liberations Top 5 Tracks of 2011	Tracks of 2011	album	event	0.0
Merry Christmas 2011	Christmas 2011	album	event	0.0
EP Review : Likes Lions - Future Colour	Likes Lions	album	album	0.0
EP Review : FaltyDL - Atlantis	FaltyDL	album	album	0.0
EP Review : Sea Oleena - Sleeplessness	Sea Oleena	album	album	0.0
EP Review : Alphabet Backwards - British Explorer	British Explorer	album	album	0.0
EP Review : Daughter - The Wild Youth	Wild Youth	album	album	0.0
New Video : Gabrielle Aplin - Home	Home	album	new	0.0
The Blog Sound of 2012	Blog Sound of 2012	event	event	0.0
Cutting Noise Podcast 002 (November 2011)	Cutting Noise	album	event	0.0
Album Review : Hectic Zeniths - Hectic Zeniths	Hectic Zeniths	album	album	0.0
New Music : Rosa Valle / Therapist / Smother Party	Therapist	album	new	0.0
New Video : The Producers - Episode 2: The Bullitts	The Producers	album	new	0.0
Album Review : Evidence - Cats & Dogs	Evidence	album	album	0.0

CProb is notably low, indicating low confidence in the guess. My believe is that since the blog is a music blog, and the titles are typically band or album names, the classifier can't distinguish titles. I initially went with this blog since it already broke down the classifications for me, but realized upon identifying the features it wouldn't be great since the classications were the only unique data. That would be my next step.

See program given in Appendix B and C.

Q3.

Assess the performance of your classifier in each of your categories by computing precision and recall. Note that the definitions are slightly different in the context of classification.

	Precision	Recall
album	19/34	19/19
new	1/1	1/5
event	10/10	10/22
single	3/3	3/4
interview	0/2	0/0

Appendix A

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

DEFAULT_COUNT=100
DEFAULT_SOURCES='http://musicliberation.blogspot.com/search?max-results=100','http://musicl
if len(sys.argv) != 3:
    print('Pass the count, defaulting to ' + str(DEFAULT_COUNT))
    print('Pass the blog, defaulting to ' + str(DEFAULT_SOURCES))
    count=DEFAULT_COUNT
    urls=DEFAULT_SOURCES
else:
    count=sys.argv[1]
    urls=sys.argv[2]

def getwords(doc):
  splitter=re.compile('\\W*')
  doc=re.compile(r'<[^>]+>').sub('',doc)
  words=[s.lower() for s in splitter.split(doc)
         if len(s)>2 and len(s)<20]
  return dict([(w,1) for w in words])

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

def getEntries(max,urls):
  entries=[]
  for url in urls:
    soup=parse(url)
    for post in soup.findAll('div',{'class':'post hentry'}):
      for entry in post.findAll('h3'):
        try:
          title=str(list(list(entry.children)[1].children)[0])
          body=''
          for bodyEle in post.findAll('div',{'style':'text-align: justify;'}):
            body+=str(bodyEle)
          if 'Reviewed by' not in title and title.strip() != '' and title not in entries:
            entries.append((title,body,),)
            if max <= len(entries):
              return entries
        except:
          print('Error on entry: ' + str(entry))
  return entries

entries=getEntries(100,urls)
print('Found ' + str(len(entries)) + ' entries. Contents:\n' + str(entries))
with open('titles.txt','w') as f:
```

```
for title,body in entries:
  f.write(title + '|' + body.replace('\n','').replace('|','') + '||\n')
```

Appendix B

```
import docclass
import os
import sys
import traceback

DATABASE_NAME='sqlite.db'

def read(classifier,titleClassification):
  i=1
  predictionList=[]
  for title,body,feature,category in titleClassification:
    try:
      predictedCategory=str(classifier.classify(body))
    except:
      print('Error calculating ' + title)
      traceback.print_exc()
      sys.exit(0)
    if(i <= len(titleClassification)/2):
      classifier.train(body, category)
      cp=-1
    else:
      cp=round(classifier.cprob(feature,predictedCategory),3)
    print('#' + str(i) + ' Title: ' + title + ' Feature: ' + feature +
          ' Predicted: ' + predictedCategory +
          ' Actual: ' + category + ' CProb: ' + str(cp))
    predictionList.append((title, feature, predictedCategory, category, cp),)
    i+=1
  return predictionList

def tex(string):
  return '\\tiny ' + string.replace('&','\&').replace('#','\#')

def writeClassificationResultsLatex(predictionList):
  with open('classificationResults.tex', 'w') as f:
    f.write('\\begin{tabular}{ l l l l l }\n')
    f.write('\\tiny Title & \\tiny Feature & \\tiny Predicted & \\tiny Category & \\tiny CP
    for title, feature, predictedCategory, category, cp in predictionList:
      f.write(tex(title) + '&' + tex(feature) + '&' + tex(predictedCategory) +
              '&' + tex(category) + '&' + tex(str(cp)) + '\\\\ \n')
    f.write('\\end{tabular}\n')

if __name__ == '__main__':
  try:
    os.remove(DATABASE_NAME)
  except:
    pass
  classifier=docclass.fisherclassifier(docclass.getwords)
  classifier.setdb(DATABASE_NAME)
  titleClassificationDict=[]
  with open('titles.txt') as f:
    for entry in f:
      try:
        title,body,feature,category=entry.split('|')
        title=title.strip().replace("'",'')
        titleClassificationDict.append((title,body,feature.strip(),
```

```
                                        category.strip()),)
      except:
        print('Error parsing entry: ' + str(entry))
  predictionList=read(classifier,titleClassificationDict)
  writeClassificationResultsLatex(predictionList)
```

Appendix C

```
#from pysqlite2 import dbapi2 as sqlite
from sqlite3 import dbapi2 as sqlite
import re
import math

def getwords(doc):
  splitter=re.compile('\\W*')
  ## Remove all the HTML tags
  doc=re.compile(r'<[^>]+>').sub('',doc)
  # Split the words by non-alpha characters
  words=[s.lower() for s in splitter.split(doc)
         if len(s)>2 and len(s)<20]

  # Return the unique set of words only
  return dict([(w,1) for w in words])

class classifier:
  def __init__(self,getfeatures,filename=None):
    # Counts of feature/category combinations
    self.fc={}
    # Counts of documents in each category
    self.cc={}
    ## extract features for classification
    self.getfeatures=getfeatures

  def setdb(self,dbfile):
    self.con=sqlite.connect(dbfile)
    self.con.execute('create table if not exists fc(feature,category,count)')
    self.con.execute('create table if not exists cc(category,count)')

  ## Increase the count of a feature/category pair
  def incf(self,f,cat):
    count=self.fcount(f,cat)
    if count==0:
      self.con.execute("insert into fc values ('%s','%s',1)"
                       % (f,cat))
    else:
      self.con.execute(
        "update fc set count=%d where feature='%s' and category='%s'"
        % (count+1,f,cat))

  ## The number of times a feature has appeared in a category
  def fcount(self,f,cat):
    res=self.con.execute(
      'select count from fc where feature="%s" and category="%s"'
      %(f,cat)).fetchone()
    if res==None: return 0
    else: return float(res[0])

  ## Increase the count of a category
```

```
def incc(self,cat):
  count=self.catcount(cat)
  if count==0:
    self.con.execute("insert into cc values ('%s',1)" % (cat))
  else:
    self.con.execute("update cc set count=%d where category='%s'"
                     % (count+1,cat))

## The number of items in a category
def catcount(self,cat):
  res=self.con.execute('select count from cc where category="%s"'
                       %(cat)).fetchone()
  if res==None: return 0
  else: return float(res[0])

## The list of all categories
def categories(self):
  cur=self.con.execute('select category from cc');
  return [d[0] for d in cur]

## The total number of items
def totalcount(self):
  res=self.con.execute('select sum(count) from cc').fetchone();
  if res==None: return 0
  return res[0]


## The train method takes an item(document) and a classification.
## It uses the getfeatures function to the break the item into its
## separate features. It then calls incf to increase the counts for
## this classification for every feature. Finally, it increases
## the total count for this classification.
def train(self,item,cat):
  features=self.getfeatures(item)
  # Increment the count for every feature with this category
  for f in features:
    self.incf(f,cat)

  # Increment the count for this category
  self.incc(cat)
  self.con.commit()

## Probability is a number between 0 and 1, indicating
## the likelihood of an event. You calculate the probability of
## a word in a particular category by dividing the number of
## times the word appears in a document in that category
## by the total number of documents in the category.
def fprob(self,f,cat):
  if self.catcount(cat)==0: return 0

  # The total number of times this feature appeared in this
  # category divided by the total number of items in this category
  return self.fcount(f,cat)/self.catcount(cat)

def weightedprob(self,f,cat,prf,weight=1.0,ap=0.5):
  # Calculate current probability
```

```
    basicprob=prf(f,cat)

    # Count the number of times this feature has appeared in
    # all categories
    totals=sum([self.fcount(f,c) for c in self.categories()])

    # Calculate the weighted average
    bp=((weight*ap)+(totals*basicprob))/(weight+totals)
    return bp



class naivebayes(classifier):

  def __init__(self,getfeatures):
    classifier.__init__(self,getfeatures)
    self.thresholds={}

  def docprob(self,item,cat):
    features=self.getfeatures(item)

    # Multiply the probabilities of all the features together
    p=1
    for f in features: p*=self.weightedprob(f,cat,self.fprob)
    return p

  def prob(self,item,cat):
    catprob=self.catcount(cat)/self.totalcount()
    docprob=self.docprob(item,cat)
    return docprob*catprob

  def setthreshold(self,cat,t):
    self.thresholds[cat]=t

  def getthreshold(self,cat):
    if cat not in self.thresholds: return 1.0
    return self.thresholds[cat]

  def classify(self,item,default=None):
    probs={}
    # Find the category with the highest probability
    max=0.0
    for cat in self.categories():
      probs[cat]=self.prob(item,cat)
      if probs[cat]>max:
        max=probs[cat]
        best=cat

    # Make sure the probability exceeds threshold*next best
    for cat in probs:
      if cat==best: continue
      if probs[cat]*self.getthreshold(best)>probs[best]: return default
    return best

## This function will return the probability that an item with the
```

```
## specified feature belongs in the specified category, assuming there
## will be an equal number of items in each category.
class fisherclassifier(classifier):
  def cprob(self,f,cat):
    # The frequency of this feature in this category
    clf=self.fprob(f,cat)
    if clf==0: return 0

    # The frequency of this feature in all the categories
    freqsum=sum([self.fprob(f,c) for c in self.categories()])

    # The probability is the frequency in this category divided by
    # the overall frequency
    p=clf/(freqsum)

    return p


  def fisherprob(self,item,cat):
    # Multiply all the probabilities together
    p=1
    features=self.getfeatures(item)
    for f in features:
      p*=(self.weightedprob(f,cat,self.cprob))

    # Take the natural log and multiply by -2
    fscore=-2*math.log(p)

    # Use the inverse chi2 function to get a probability
    return self.invchi2(fscore,len(features)*2)

  ## Inverse chi-squared function
  def invchi2(self,chi, df):
    m = chi / 2.0
    sum = term = math.exp(-m)
    for i in range(1, df//2):
        term *= m / i
        sum += term
    return min(sum, 1.0)

  def __init__(self,getfeatures):
    classifier.__init__(self,getfeatures)
    self.minimums={}

  def setminimum(self,cat,min):
    self.minimums[cat]=min

  def getminimum(self,cat):
    if cat not in self.minimums: return 0
    return self.minimums[cat]

  def classify(self,item,default=None):
    # Loop through looking for the best result
    best=default
    max=0.0
    for c in self.categories():
```

```
    try:
      p=self.fisherprob(item,c)
    except:
      print('Error on item: ' + str(item) + ' category: ' + c)
      continue
    # Make sure it exceeds its minimum
    if p>self.getminimum(c) and p>max:
      best=c
      max=p
  return best
```