Web Science cs532-s16

Assignment 9 Report

Dr. Michael L. Nelson

By: Huan Huang
04/22/2016

Problem 1

```
Choose a blog or a newsfeed (or something similar with an Atom
or RSS feed).  Every student should do a unique feed, so please
"claim" the feed on the class email list (first come, first served).
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 (or items if RSS).

Create between four and eight different categories for the entries
in the feed:

examples:

work, class, family, news, deals

liberal, conservative, moderate, libertarian

sports, local, financial, national, international, entertainment

metal, electronic, ambient, folk, hip-hop, pop

Download and process the pages of the feed as per the week 12
class slides.
```

Answer

I took me a long time to find a web page with over 100 RSS feeds. I eventually used PlayStation's RSS feeds for movies, which has over 10 thousand feeds. I downloaded the feeds with the curl -o command and saved it as psMovies.xml. Next, I picked out the categories for classifying the entries, which are:

action - a film genre in which the characters are thrust into a series of challenges that involve physical feats, extended fight scenes, violence, and frantic chases.

comedy - a genre of film in which the main emphasis is on humour. These films are designed to make the audience laugh through amusement and most often work by exaggerating characteristics for humorous effect. Films in this style traditionally have a happy ending (black comedy being an exception).

scifi - a film genre that uses science fiction: speculative, fictional science-based depictions of phenomena that are not fully accepted by mainstream science, such as extraterrestrial life forms, alien worlds, extrasensory perception and time travel, along with futuristic elements such as spacecraft, robots, cyborgs, interstellar space travel or other technologies. Science fiction films have often been used to focus on political or social issues, and to explore philosophical issues like the human condition.

horror - unsettling films designed to frighten and panic, cause dread and alarm, and to invoke our hidden worst fears, often in a terrifying, shocking finale, while captivating and entertaining us at the same time in a cathartic experience.

romance - romantic love stories recorded in visual media for broadcast in theaters and on television that focus on passion, emotion, and the affectionate romantic involvement of the main characters and the journey that their genuinely strong, true and pure romantic love takes them through dating, courtship or marrige. Romance films make the romantic love story or the search for strong and pure love and romance the main plot focus.

other - other movie genres that are not included above.
(The movie genre definitions are from WIKIPEKIA)

Problem 2

```
Manually classify the first 50 entries, and then classify (using
the fisher classifier) the remaining 50 entries.

Create a table with the title, predicted category, actual category,
and cprob() and fisherprob() for the actual category.
```

Answer

I copy and modified the files docclass.py and feedfilter.py to solve this problem. The program feedfilter.py reads and parse the entries from psMovies.xml. And for each of the top 50 entries, I enter an actual classification which is use to call the classifier.train() function in docclass.py. This is used to train the classifier and enables it to make predictions of the category of the movies based on my input. I also get the fisher's conditional probability by calling the classifier.cprob() and classifier.fisherprob() functions, the actual category is to those functions to calculate the probabilities .

-------------------------------- Manually classify first 50 entries --------------------------------

Title	Prediction	Actual	cprob	fisherprob
Holy Ghost People	None	other	1.0	0.75
Bad Ass 2: Bad Asses	other	action	0.0	0.8861
Dom Hemingway - Theatrical Tralier	other	comedy	0.0	0.75
Nymphomaniac: Volume 2	other	other	1.0	0.75
In The Blood	action	action	1.0	0.75
10 Rules for Sleeping Around	action	romance	1.0	0.75
The Unknown Known	action	other	1.0	0.75
Afflicted	action	horror	1.0	0.75
Alien Abduction	horror	horror	0.0	0.8861
Alan Partridge	action	comedy	1.0	0.75
The Occupants	action	horror	1.0	0.75
The Little Rascals Save the Day	action	comedy	1.0	0.75
Seal Team Eight: Behind Enemy Lines	horror	action	1.0	0.75
The Pirate Fairy	horror	other	1.0	0.75
The Pirate Fairy (3D)	other	other	1.0	0.8333
Ages and Stages: The Story of the Meligrove Band	other	other	1.0	0.875
Blumenthal	action	comedy	1.0	0.75
Let Timmy Smoke	other	other	0.0	0.8861
I Am Divine	action	other	1.0	0.75
Four Seasons	horror	romance	1.0	0.75
About A Zombie	action	horror	0.0	0.7428
The Living Matrix	other	other	1.0	0.8333
Herbert Nitsch: Back from the Abyss	comedy	other	1.0	0.75
HazMat	horror	horror	1.0	0.75
Battle of the Empires	action	action	1.0	0.75
Volcom: True to This	horror	other	0.0	0.8861
E.T.X.R.	horror	scifi	0.0	0.8738
Hide Your Smiling Faces	horror	other	1.0	0.75
Locker 13	other	other	1.0	0.75
Airplane vs Volcano	horror	scifi	1.0	0.75
Mistaken For Strangers	other	comedy	1.0	0.75
Red Bull X Fighters 2013	horror	other	0.2632	0.3421
The Appearing	horror	horror	0.0	0.8861
SSI: Sex Squad Investigation	other	comedy	1.0	0.75
Black Roots	horror	other	0.0	0.8861
Happy Camp	horror	horror	1.0	0.75
Odd Thomas	horror	comedy	0.0	0.8861
Countdown	horror	horror	1.0	0.75
Awake Zion	horror	other	1.0	0.75
Beast of the Bering Sea	other	scifi	1.0	0.75
Wendy Liebman: Taller on TV	horror	comedy	0.0	0.9188
How to Follow Strangers	comedy	romance	1.0	0.75
In Heaven There is No Beer	horror	other	0.4848	0.4886
Come Back, Africa	horror	other	0.3077	0.3718
Home	horror	other	1.0	0.75
Scooby-Doo! Wrestlemania Mystery	comedy	comedy	0.0	0.8861
Avengers Confidential: Black Widow & Punisher	comedy	scifi	1.0	0.75
Bad Words - Theatrical Tralier	comedy	comedy	0.0	0.6207
Cheap Thrills	horror	horror	1.0	0.75
The French Minister	comedy	comedy	0.0	0.8861

---------------------------- Other 50 automaticly classified entries ------------------------------

Title	Prediction	Actual	cprob	fisherprob
Mardock Scramble: Third Exhaust	other	action	0.0	0.5
Blood Ties (2014)	horror	other	0.0	0.25
Maladies	comedy	other	0.0	0.5
Divergent - Theatrical Trailer	comedy	scifi	0.0	0.5
McCanick	horror	action	0.0	0.5
Nymphomaniac: Volume 1	other	other	0.0	0.7428
Battle of the Undead	horror	horror	0.0	0.5
American Virgins	horror	comedy	0.0	0.0714
Almost Sharkproof	horror	comedy	0.0	0.5
Sparks	horror	action	0.0	0.5
The Wrath of Vajra	action	action	0.0	0.5399
The Laughing Matter	horror	comedy	1.0	0.8333
Tom Hollands Twisted Tales	horror	horror	0.0	0.5
Jeff Dunhams Achmed Saves America: The Animated Movie	horror	comedy	0.0	0.4752
Waking	horror	romance	0.0	0.5
Romeo, Romeo	horror	romance	0.0	0.5
CyberGeddon	horror	other	0.0	0.5
Signature Sounds: Music of WWE	horror	other	0.0	0.25
Maidentrip	romance	other	0.0	0.5
Foxfire	romance	other	0.0	0.1
Danny MacAskills Imaginate Documentary	comedy	other	0.0	0.5
Buck Wild	comedy	comedy	0.0	0.5
5th Street	horror	action	0.0	0.5
How to Be a Man	other	comedy	0.0	0.3997
UFC Event Replays | UFC 171: Hendricks vs. Lawler	horror	other	0.0	0.5
Patrick: Evil Awakens	horror	horror	0.0	0.5
Better Living Through Chemistry	horror	comedy	0.0	0.2358
The Den	horror	horror	0.0	0.5
The Art of the Steal	horror	comedy	0.0	0.5
uwantme2killhim?	horror	horror	0.0	0.5
The Right Kind of Wrong	other	romance	0.0	0.5
Need for Speed - Theatrical Trailer	comedy	action	0.0	0.5
Veronica Mars	horror	comedy	0.0	0.5
The Last Days	scifi	scifi	0.0	0.5
WWE WCWs Greatest PPV Matches Vol. 1	other	other	0.0	0.5966
Bubble Guppies: Animals Everywhere!	other	other	0.0	0.3964
End of the World	comedy	scifi	0.0	0.5
A Cross to Bear	horror	other	0.0	0.5966
Shadowboxing	horror	comedy	0.0	0.5
Dark House	horror	horror	0.0	0.5
The Hungover Games	other	comedy	0.0	0.5
The Hungover Games (Unrated)	other	comedy	0.0	0.5
A Dance for Bethany	horror	romance	0.0	0.5
WWE WCWs Greatest PPV Matches Vol. 3	other	other	0.0	0.5966
WWE WCWs Greatest PPV Matches Vol. 2	other	other	0.0	0.5966
Soap Life	other	other	0.0	0.5966
Hated	other	other	0.6346	0.6077
In the Name of the King 3: The Last Mission	other	action	0.0	0.5
Celebrating the Music of "Inside Llewyn Davis"	other	other	0.3426	0.374
All Things to All Men	horror	other	0.0	0.1667

```python
import feedparser
import re
import math
import docclass

savefile = open("data.txt", "a")

# Takes a filename or URL of a blog feed and classifies the entries
def read(feed,classifier):
```

```

splitRegexp = re.compile( r"<[^>]+>" )

num=0
# Get feed entries and loop over them
f=feedparser.parse(feed)

string1 = '-------------------------------- Manually classify first 50 entries --------------------------------\n'
dividline = "_" * 100 + "\n"
print string1
savefile.write(string1)
string2 = "{:<60}{:<12}{:<10}{:<8}{:<10}{}".format("Title", "Prediction", "Actual", 'cprob', 'fisherprob',"\n")
print string2
savefile.write(string2)
savefile.write(dividline)

for entry in f['entries'][0:50]:
  num=num +1
  # Print the contents of the entry
  title=entry['title'].encode('utf-8').replace("'","")
  print 'Title:     '+ title

  summary = splitRegexp.sub( "", entry[ "summary" ] )

  print summary #entry['summary'].encode('utf-8')

  # Combine all the text to create one item for the classifier
  #fulltext='%s\n%s\n%s' % (entry['title'],entry['publisher'],entry['summary'])
  fulltext='%s\n%s' % (entry['title'],entry['summary'])
  # Remove apostrophes
  fulltext=fulltext.replace("'","")
  # Print the best guess at the current category
  predicted=str(classifier.classify(fulltext))
  print 'Predicted category: ', predicted

  # Ask the user to specify the correct category and train on that
  actual=raw_input('Actual category: ')
  classifier.train(fulltext, actual)

  feature=raw_input('Enter string classifier: ')

  #classifier.train(entry,cl)
  # probability the item should be in this category
  cp=round(classifier.cprob(feature,predicted),4)
  fcp=round(classifier.fisherprob(feature,predicted),4)
  print 'cprob: ', str(cp)
  print 'fisherprob: ', str(fcp)
  string3 = "{:<60}{:<12}{:<10}{:<8}{:<10}{}".format(title, predicted, actual, str(cp), str(fcp), "\n")
  savefile.write(string3)

```

```
    # Save the manual classifications
    # num, entry, feature, predicted, actual, cprob=None
    classifier.manualClassdb(num, title, feature, predicted, actual, cp, fcp)
  savefile.write(dividline)
#def autoClassify(feed,classifier):
  num=50
  string4 = '\n---------------------------------------- Other 50 automaticly classified entries ----------------------------------------\n'
  print string4
  savefile.write(string4)
  savefile.write(string2)
  savefile.write(dividline)
  # Get feed entries and loop over them
  #f=feedparser.parse(feed)
  for entry in f['entries'][50:100]:
    num=num+1
    # Print the contents of the entry
    title=entry['title'].encode('utf-8').replace("'","")
    print 'Title:      '+ title
    summary = splitRegexp.sub( "", entry[ "summary" ] )

    print summary #entry['summary'].encode('utf-8')

    # Combine all the text to create one item for the classifier
    #fulltext='%s\n%s\n%s' % (entry['title'],entry['publisher'],entry['summary'])
    fulltext='%s\n%s' % (entry['title'],entry['summary'])
    fulltext=fulltext.replace("'","")
    # Print the best guess at the current category
    predicted=str(classifier.classify(fulltext))
    print 'Predicted: ', predicted

    # Ask the user to specify the correct category
    actual=raw_input('Actual category: ')
    feature=raw_input('Enter string classifier: ')


    # probability the item should be in this category
    cp=round(classifier.cprob(feature,predicted),4)
    fcp=round(classifier.fisherprob(feature,predicted),4)
    print 'cprob: ', str(cp)
    print 'fisherprob: ', str(fcp)

    string6 = "{:<60}{:<12}{:<10}{:<8}{:<10}{}".format(title, predicted, actual, str(cp), str(fcp), "\n")
    savefile.write(string6)
    # Save the trained classifications
    # num, entry, feature, predicted, cprob(feature, predicted)
    classifier.autoClassdb(num, title, feature, predicted, actual, cp, fcp)
  #return classifier
  savefile.write(dividline)

def entryfeatures(entry):
  splitter=re.compile('\\W*')
```

```
  f={}

  # Extract the title words and annotate
  titlewords=[s.lower() for s in splitter.split(entry['title'])
          if len(s)>2 and len(s)<20]
  for w in titlewords: f['Title:'+w]=1

  # Extract the summary words
  summarywords=[s.lower() for s in splitter.split(entry['summary'])
          if len(s)>2 and len(s)<20]

  # Count uppercase words
  uc=0
  for i in range(len(summarywords)):
    w=summarywords[i]
    f[w]=1
    if w.isupper(): uc+=1

    # Get word pairs in summary as features
    if i<len(summarywords)-1:
      twowords=' '.join(summarywords[i:i+1])
      f[twowords]=1

  # Removed: Keep creator and publisher whole
  #f['Publisher:'+entry['publisher']]=1

  # UPPERCASE is a virtual word flagging too much shouting
  if float(uc)/len(summarywords)>0.3: f['UPPERCASE']=1

  return f


cl=docclass.fisherclassifier(docclass.getwords)
cl.setdb('psMovies.db')
read('psMovies.xml',cl)
```

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

def getwords(doc):
  splitter=re.compile('\\W*')
  #print doc
  ## 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 rss(num, entry, feature, predicted, actual, cprob)')
    self.con.execute('create table if not exists fc(feature,category,count)')
    self.con.execute('create table if not exists cc(category,count)')
    # remove old data from previous sessions
    self.con.execute('delete from rss')
    self.con.execute('delete from fc')
    self.con.execute('delete from cc')

  def manualClassdb (self, num, entry, feature, predicted, actual, cp, fcp):
    self.con.execute("insert into rss values ('%s', '%s', '%s', '%s', '%s', '%s')"
                     % (num, entry, feature, predicted, actual, fcp))
    self.con.commit()

  def autoClassdb (self, num, entry, feature, predicted, actual, cp, fcp):
    self.con.execute("insert into rss values ('%s', '%s', '%s', '%s', '%s', '%s')"
                     % (num, entry, feature, predicted, actual, fcp))
    self.con.commit()
  ## 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.lower()))
    else:
      self.con.execute(
        "update fc set count=%d where feature='%s' and category='%s'"
        % (count+1,f,cat.lower()))

  ## 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.lower()))
```

```
    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():
      p=self.fisherprob(item,c)
      # Make sure it exceeds its minimum
      if p>self.getminimum(c) and p>max:
        best=c
        max=p
    return best
```

Problem 3

Assess the performance of your classifier in each of your categories by computing precision, recall, and F-measure.

Answer

In order to calculate the precision, recall, and F-measures. I first have to get the false positive, false negative, and true positive of the predictions. Where the prediction matched my actual input for each category is the true positive for each category. The false positive and false negative is obtained by subtracting the TP from predicted and actual counts of each category. Then use the formulas from lecture slides to calculated the precision, recall, and F-measures.

Precision = TP / (TP+FP)
Recall = TP / (TP+FN)
F-Measure = 2 * P*R / (P+R)

Categories	FP	FN	TP	Precision	Recall	F-measure
Action	10	7	3	0.23	0.3	0.26
Comedy	8	19	4	0.33	0.17	0.22
Horror	35	3	12	0.26	0.8	0.39
Other	10	24	14	0.58	0.37	0.45
Romance	2	7	0	0	0	0
Scifi	0	6	1	1	0.14	0.25

Based on the result, the classifier did not perform well at all, but it was expected, since I only trained it with 50 entries.