CS896 Introduction to Web Science
Fall 2013
Report for Assignment 10

Corren G. McCoy

December 12, 2013

Contents

List of Tables

1 Question 1

1.1 Problem

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 pages of the feed as per the week 12 class slides.

1.2 Response

For our blog, we chose Jingle Bell Junction (`http://jinglebelljunction.com/`), which is self-described as "the merriest Christmas site on the web!" The content of the blogs and d downloads consist of holiday recipes, news & articles, crafts, homemade gift ideas, and various other Christmas-related topics. After reviewing the blogs, we chose the following categories:

- recipe;
- craft;
- activity;
- gift idea; and
- news article.

To ensure consistency as we are attempting to classify the entries, the RSS feed was downloaded to an XML file (i.e., jinglebell.xml). The XML file is included in the supporting documents for this assignment.

2 Question 2

2.1 Problem

Manually classify the first 50 entries, and then classify (using the fisher classifier) the remaining 50 entries. Report the cprob() values for the 50 titles as well. From the title or entry itself, specify the 1-, 2-, or 3-gram that you used for the string to classify. Do not repeat strings; you will have 50 unique strings. Create a table with the title, the string used for classification, cprob(), predicted category, and actual category.

2.2 Response

We used the docclass.py and feedfilter.py files found in Segaran [1] as the basis for our document filtering. The modified source code, shown in Appendices A and B, performs the following tasks:

- Use *doclass.py* to extract features from the title and summary from the first 50 entries in the RSS feed.
- Remove HTML tags before dividing the remaining text of each entry into individual words.
- Manually train the classifier using the Fisher method. Save the features and related categories to a set of database tables so the training will persist between sessions.
- Use *feedfilter.py* to parse the title and summary of the RSS feed for the remaining 50 entries.

- Use the Fisher method to predict a category based on the entry.
- Prompt the user to enter the actual category along with an n-gram to determine the Fisher probability (i.e. cprob()).
- Save the entry title, feature, predicted category, actual category, and cprob() to database table.

Table 1 shows the number of entries that were allocated to each of defined categories while training the classifier.

Category	Entries
craft	15
recipe	4
activity	11
news	24
gift	6

Table 1: Entries per Training Category

The results of classifying the Jinglebell Junction RSS feed are shown in Table 2. Based on the overwhelming number of probabilites rated at 0, we can determine the classifier did not perform well on this particular data. As stated in Segaran [1], this might be attributed to the unequal number of documents allocated to each category during training. We can see from Table 1 that blogs in Jinglebell Junction skew more heavily towards news articles and craft ideas.

3 Question 3

3.1 Problem

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; see: `http://en.wikipedia.org/wiki/Precision_and_recall#Definition_.28classification_context.29`

3.2 Response

Since our data is based on a multiple classification, we will use a confusion matrix (`http://en.wikipedia.org/wiki/Confusion_matrix`) to analyze the results and calculate precision and recall. The matrix as shown in Table 3 will illustrate how well the classifier was able to make the correct predictions. Based on the entries in the confusion matrix, we can now compute precision and recall as shown in Table 4. We can see that entries in the *recipe* category have the highest level of precision, while entries in the *news* category have the highest level of recall.

4 Question 4 Extra Credit

4.1 Problem

Redo the questions above, but with the extensions on slide 26 and pp. 136–138.

4.2 Response

Not attempted.

Title	**Feature**	**cprob()**	**Predicted**	**Actual**
Poinsettia Fact and Fiction	belief	1.0	news	news
Reindeer Hokey Pokey	hokey pokey	0.0	news	activity
Christmas Punch	punch	0.0	craft	recipe
Holiday Pumpkin Cheesecake	cheesecake	0.0	craft	recipe
Chocolate Chip Toffee M&M Cookies	cookies	0.0	news	recipe
Pumpkin Walnut Fudge	fudge	0.0	recipe	recipe
Peanut Brittle	peanut	0.0	gift	recipe
Peanut Butter Cups	butter	0.0	news	recipe
Double Chocolate Caramels	chocolate	0.5	recipe	recipe
Easiest Fudge in the World!	ingredient	0.0	recipe	recipe
Jinglebelles Pumpkin Pancakes	pancake	0.0	recipe	recipe
Spiced Pumpkin Fudge	spiced	0.0	recipe	recipe
Jinglebelles Chocolate Ice Cream	ice cream	0.0	news	recipe
Hot Russian Tea	tea	0.0	news	recipe
Scented Gel Air Fresheners	scented	0.0	activity	craft
Paint Stirrer Snowman	paint	0.0	activity	craft
Craft Stick Angel	stick	0.0	craft	craft
The Origins of Santa	santa	0.234	news	news
Snowman Soup Hot Chocolate	soup	0.0	recipe	recipe
Decorated Clay Ornaments	clay	0.0	news	craft
Christmas Sponge Art	sponge	0.0	recipe	craft
Reindeer Candycane Ornament	ornament	0.0	recipe	activity
Chocolate Melting Spoons	spoons	0.0	recipe	craft
How to find your screen resoultion	screen	0.0	news	news
Installing Christmas wallpapers on your iPhone	wallpapers	0.0	news	news
Santa Hat Gift Tags	tags	0.0	gift	craft
How to cook a perfect Thanksgiving turkey	turkey	1.0	news	activity
Craft Stick Angel	craft	1.0	craft	craft
Jingle Bell Napkin Rings	napkin	0.0	craft	craft
2011 Christmas Expo Lights Up Gatlinburg, TN This Summer	expo	0.0	news	activity
“My Favorite Gift”by Virginia Blanck Moore	favorite gift	0.0	news	news
Starting & Adding to Your Christmas Music Library A Primer	music	0.0	news	activity
Simple Techniques for Keeping Your Child Believing in Santa Claus	simple techniques	0.0	news	news
Embossed Velvet Stockings	velvet	0.0	news	craft
Snowman Clip Art	clip art	0.0	news	craft
Santa Claus Clip Art	graphics	0.0	news	craft
Grinch Coloring	grinch	0.0	craft	activity
Christmas House Clip Art	house	0.125	news	craft
A Jinglebell Junction Exclusive!! Two Trees	two trees	0.0	news	activity
Jar Lid Magnets	magnets	0.0	activity	craft

Table 2: Classification Results

	Predicted Class				
Actual	recipe	craft	gift	news	activity
recipe	6	2	1	4	0
craft	2	3	1	5	3
gift	0	0	0	0	0
news	0	0	0	6	0
activity	1	1	0	5	0

Table 3: Confusion Matrix

	Precision	Recall
recipe	6/9 (0.67)	6/13 (0.46)
craft	3/6 (0.50)	3/14 (0.21)
gift	0/2 (0.00)	0/0 (0.00)
news	6/20 (0.30)	6/6 (1.00)
activity	0/3 (0.00)	0/7 (0.00)

Table 4: Performance Measures

Bibliography

[1] T. Segaran. *Programming collective intelligence: building smart web 2.0 applications*. O'Reilly Media, 2007.

Appendix A

Python Source Code - docclass.py

```
#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):
  self.con.execute("insert into rss values ('%s','%s', '%s', '%s','%s', '%s')"
                   % (num, entry, feature, predicted, actual, None))
  self.con.commit()

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

Appendix B

Python Source Code - feedfilter.py

```
import feedparser
import re
import math

# Takes a filename or URL of a blog feed and classifies the entries
def read(feed,classifier):
  num=0
  # Get feed entries and loop over them
  f=feedparser.parse(feed)
  print
  print '----- Begin manual classification (training) -----'
  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
    #print 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: ')
    feature=None
    classifier.train(fulltext, actual)

    # Save the manual classifications
    # num, entry, feature, predicted, actual, cprob=None
    classifier.manualClassdb(num, title, feature, predicted, actual)
```

```
#def autoClassify(feed,classifier):
  num=50
  print '----- Begin automatic classification -----'
  # Get feed entries and loop over them
  f=feedparser.parse(feed)
  for entry in f['entries'][51:100]:
    num=num+1
    # Print the contents of the entry
    title=entry['title'].encode('utf-8').replace("'","")
    print 'Title:     '+ title
    #print 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('Enter actual category: ')
    feature=raw_input('Enter string classifier: ')

    #classifier.train(entry,cl)
    # probability the item should be in this category
    cp=round(classifier.cprob(feature,predicted),3)
    print 'cprob: ', str(cp)
    # Save the trained classifications
    # num, entry, feature, predicted, actual, cprob(feature, predicted)
    classifier.autoClassdb(num, title, feature, predicted, actual, cp)
  #return classifier

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
```