Justin Schaffner (JASchaff)
CS532 — Spring 2017

Assignment 9

April 20, 2017

Problem 1: Build the Data Set

Question 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. Download and process the pages of the feed as per the week 12 class slides. Be sure to upload the raw data (Atom or RSS) to your github account. Create a table with 100 rows. This is your "ground truth" (or "gold standard") data.

I had a really hard time finding a blog with 100 entries that allowed the RSS feed to go beyond 10 or 20 items. Most of the pages I found don't have rel=next links for their RSS feeds. Instead I gathered several blogs that were all about cycling and built my data set from that. The list of blogs I used can be found in cyclefile.txt. I used the script getRSS.py in Listing 1 to pull the RSS pages and store them in individual files. In order to make a list of the 100 entries I intend to use, I added the function "downtohundred()", shown in Listing 2 to fishermethod.py. Fishermethod will hold all of the functions used in this assignment other than getRSS, while the main script calling the functions can be found in feedclassify.py. I created 7 different categories for the entries:
community recommendation-travel recommendation-gear recommendation-media opinion how-to news
The categories are pretty self-explanatory, except perhaps for community, which I use for personal stories and anecdotes. The first section of feedclassify.py deals with the ground truth. Once the groundtruth.txt file has been established, it won't ask for it on further iterations. feedclassify.py can be seen in Listing 4 at the end of this report along with fishermethod.py in Listing 3.

Listing 1: Q1 getRSS.py

```
import requests

header={'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10
    _12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
    /56.0.2924.87 Safari/537.36', 'Accept-Language':'en'}
URL=[]
file=open('cyclefile.txt', 'r')
for line in file:
    URL.append(line.rstrip())
count=0
for url in URL:
    page=requests.get(url, headers=header, allow_redirects=True,
        stream=True)
    out=open('feeds/RSSfeed'+str(count)+'.xml', 'w')
    out.write(page.text)
    count+=1
```

Listing 2: Q1 downtohundred

```
def downtohundred(path):
    articles={}
    for file in os.listdir(path):
        filepath=os.path.join(path, file)
        d=feedparser.parse(filepath)
        for e in d.entries:
            if 'summary' in e:
                articles.setdefault(e.title , e.summary)
            else:
                articles.setdefault(e.title , e.description)
    return [(v, k) for v,k in articles.items()][:100]
```

Problem 2: Fisher Classifier

Train the Fisher classifier on the first 50 entries (the "training set"), then use the classifier to guess the classification of the next 50 entries (the "test set"). Create a table with 50 rows. Assess the performance of your classifier in each of your categories by computing precision, recall, and F-measure. Use the "macro-averaged" label based method, as per: http://stats.stackexchange.com/questions/21551/how-to-compute-precisionrecall-for-multiclass-multilabel-classification For example, if you have 5 categories (e.g., 80s, metal, alternative, electronic, cover), you will compute precision, recall, and F-measure for each category, and then compute the average across the 5 categories.
For question 2 I split the list of 100 articles and sent them to my own version of the read() function called “myread()” in fishermethod.py. The main difference is the ability to handle files from multiple feeds. The entries in the files are compared to the list of 100 from groundtruth.txt. Once the Classifier has been trained on the first 50, it calculates its best guess for the remaining 50. The table can be seen in results0.txt or an excerpt in Table 1. For the Precision, Recall and F-measure I used the Pandas-ML Library to make a confusion matrix. The Pandas-ML library does most of the work for you and generates the TP, FP and FN automatically. Those results can be seen in Table 2 along with the Macro-Avg. Even though there was a category for news, there are apparently no news articles in the second half of the data set, so news is not represented. The results aren't terribly good. This could be because some of the entries technically fall in more than one category.

Table 1: 50/50 Results

Title	Actual	Predicted
California Basketpackin?	opinion	recommendations-gear
In Search of the Best Bike Bell: The Bikeway Shootout.	recommendations-gear	recommendations-gear
Trans Ecuador Mountain Bike Route: Dirt Road Version	recommendations-travel	recommendations-travel
Lost Coast North	recommendations-media	recommendations-media
New Santa Cruz Chameleon: Bikepacking-ready trail stud?	recommendations-gear	recommendations-gear
Salsa Warbird Review: An Underbiking truce.	recommendations-gear	recommendations-travel
Revelate Mag-Tank: Fit for Vikings	recommendations-gear	recommendations-gear
Baja Divide: A Film by Tales on Tyres	recommendations-media	recommendations-media
Trans-Cuba: La Ruta Mala	community	recommendations-travel

Table 2: 50/50 FPR

Categorie	Precision	Recall	F-measure
Community	0.20	0.14	0.17
How-to	0.00	0.00	0.00
Opinion	0.00	0.00	0.00
Recommendations-gear	0.55	0.90	0.68
Recommendations-travel	0.57	0.67	0.62
Recommendations-media	0.80	0.80	0.80
Macro-Avg	0.35	0.42	0.38

Problem 3: Fisher Classifier on 90 trainer items

Repeat question 2, but use the first 90 entries to train your classifier and the last 10 entries for testing. I redivided the original list of 100 and ran the same functions over the lists, but using a new database. The results are in results1.txt or in Table 3 and the Precision, Recall and F-measure are in Table 4

Table 3: 90/10 Results

Title	Actual	Predicted
3 Car Free Gravel Climbs in Southern California	recommendations-travel	how-to
3 Must-Do Rides in Mississippi	recommendations-travel	how-to
Moulton Review (How do the Moultons and Bromptons compare?!)	recommendations-gear	recommendations-gear
700c to 650b Conversion Project	how-to	recommendations-travel
Shop Visit: Gladys Bikes	community	community
Bicycling Across Mississippi Video Playlist	community	community
PLPTucson: Shipping Bikes with the AirCaddy Bike Box	recommendations-gear	recommendations-gear
The Gino Behind the Gino Mount	recommendations-gear	how-to
Norther Cycles ? Randonneuring Paradise	recommendations-gear	how-to
Visiting Rivelo PDX	community	how-to

Table 4: 90/10 FPR

Title	Precision	Recall	F-measure
Community	1.00	0.67	0.80
How-to	0.00	0.00	0.00
Recommendations-gear	1.00	0.50	0.67
Recommendations-travel	0.00	0.00	0.00
Macro-Avg	0.50	0.29	0.36

Listing 3: Fisher Method

```
import re
import math
import feedparser
import requests
from bs4 import BeautifulSoup as bs
from sqlite3 import dbapi2 as sqlite
import os
import pandas_ml as pd

header={'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10
    _12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
    /56.0.2924.87 Safari/537.36', 'Accept-Language':'en'}
```

```
def F_P_R(articles, classifier):
    categories=classifier.categories()
    values=list((k,v) for k,v in articles.values())
    pred=list(v for k,v in values)
    act=list(k for k,v in values)
    pred=[item.rstrip() for item in pred]
    act=[item.rstrip() for item in act]
    conf=pd.ConfusionMatrix(act,pred)
    conf.print_stats()
```

```
def downtohundred(path):
    articles={}
    for file in os.listdir(path):
        filepath=os.path.join(path, file)
        d=feedparser.parse(filepath)
        for e in d.entries:
            if 'summary' in e:
                articles.setdefault(e.title, e.summary)
            else:
                articles.setdefault(e.title, e.description)
    return [(v, k) for v,k in articles.items()][:100]
```

```
# Takes a filename of URL of a blog feed and classifies the entries
def read(feed,classifier):
    # Get feed entries and loop over them
    f=feedparser.parse(feed)
    for entry in f.entries:
        if 'summary' in entry:
            text=entry.summary
        else:
            text=entry.description
        print('\n------')
        # Print the contents of the entry
        print('Title: '+entry.title.encode('utf-8'))
        print('Published: '+entry.published.encode('utf-8'))
        print('\n'+text.encode('utf-8'))
        # Combine all the text to create one item for the
            classifier
        fulltext='%s\n%s\n%s' % (entry.title,entry.published,text)
        # Print the best guess at the current category
        print('Guess: '+str(classifier.classify(fulltext)))
        # Ask the user to specify the correct category and train on
            that
        cl=raw_input('Enter category: ')
        classifier.train(fulltext,cl)


#my take on the read function. takes a list of titles to be found
    in a file of feeds
def myread(articles, path, classifier):
    entries=[]
    for file in os.listdir(path):
        feed=os.path.join(path, file)
        f=feedparser.parse(feed)
        for entry in f.entries:
            entries.append(entry)

    for entry in entries:
        if entry.title in articles.keys():
            if 'summary' in entry:
                text=entry.summary
            else:
                text=entry.description
            print('\n-----')
            # Print the contents of the entry
            print('Title: '+ str(entry.title))
            print('\n'+str(text))
            # Combine all the text to create one item for the
                classifier
            fulltext='%s\n%s' % (entry.title,text)
            # Print the best guess at the current category
            print('Guess: '+str(classifier.classify(fulltext)))
            # Ask the user to specify the correct category and
                train on that
            cl=input('Enter category: ')
            classifier.train(fulltext,cl)


def myclassify(articles, path, classifier):
    entries=[]
    for file in os.listdir(path):
        feed=os.path.join(path, file)
        f=feedparser.parse(feed)
        for entry in f.entries:
            entries.append(entry)

    for entry in entries:
        if entry.title in articles.keys():
            if 'summary' in entry:
                text=entry.summary
            else:
                text=entry.description
            fulltext='%s\n%s' % (entry.title,text)
            best=classifier.classify(fulltext)
            articles[entry.title]=(articles[entry.title], best)

    if not os.path.exists('./results'):
        os.mkdirs('./results')
    file=os.path.join('./results/','results'.rstrip()+str(len(os.
       listdir('./results')))+'.txt')
    with open(file, 'w') as out:
        for key in articles.keys():
            print(key, *articles[key], sep='\t', end='\n', file=out
                )
    F_P_R(articles, classifier)

def getwords(doc):
    splitter=re.compile('\\W*')
    # 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={}
        self.getfeatures=getfeatures

    # 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))

# 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 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])

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

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( )

    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

    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)')

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

    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)

    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 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
```

Listing 4: Feed Classify

```
import requests
from fishermethod import *
import feedparser
import os

articles={}
#establish ground truth, if not already done
if not os.path.isfile('./groundtruth.txt'):
    listarticles=downtohundred('./feeds')
    with open('groundtruth.txt', 'w') as gtout:
        print('Establishing Ground Truth: %d'% len(articles))
        for a in listarticles:
            print(a[0], a[1], sep=': ')
            cat=input('Enter category: ')
            print(a[0], cat, sep='\t', file=gtout)
            articles.setdefault(a[0],cat)

else:
    for line in open('groundtruth.txt', 'r'):
        (title, actual)=line.split('\t')
        articles.setdefault(title,actual)

#process for question 2
halfone=dict(list(articles.items())[:50])
halftwo=dict(list(x for x in articles.items() if x not in halfone.
   items()))
cl=fisherclassifier(getwords)
if not os.path.isfile('./cyclingdb.db'):
    cl.setdb('cyclingdb.db')
    myread(halfone, './feeds', cl)
else: cl.setdb('cyclingdb.db')
myclassify(halftwo, './feeds', cl)

#process for question 3
cl2=fisherclassifier(getwords)
halfninety=dict(list(articles.items())[:90])
halften=dict(list(x for x in articles.items() if x not in
   halfninety.items()))
if not os.path.isfile('./cyclingdb2.db'):
    cl2.setdb('cyclingdb2.db')
    myread(halfninety, './feeds', cl2)
else:cl2.setdb('cyclingdb2.db')
myclassify(halften, './feeds', cl2)
```