Bo Zhang
01063214

1 Create "Ground Truth" Data

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. Create a table with 100 rows of entries' names and categories.

Algorithm:

1. Open the feeds 1 by 1.
2. Get the title and summary of each different entry, use the feed's title as the category, then save the first 100 entries' titles and their categories to "entries.txt" and save the first 100 summaries to "summary.txt".

Source code:

Listing 1: The content of categoriesSplit.py

```
import feedparser
import re

entries={}
out=open('entries.txt','w',encoding='utf-8')
out.write('title\tclassification\n')
feedlist=[line.strip() for line in open('feedlist.txt','r')]
for feedurl in feedlist:
    print (feedurl)

    d=feedparser.parse(feedurl)
    print (d.feed.title)

    # Loop over all the entries
    for e in d.entries:
        print (e.title)

        summary=''
        if 'content' in e:
            for content in e.content:
                print (content)
                print ()
                summary=summary+content.value+' '
        elif 'summary' in e and e.summary!='':
            print (e.summary)
```

```
            print ()
            summary=summary+e.summary+' '
        elif 'description' in e and e.description!='':
            print (e.description)
            print ()
            summary=summary+e.description+' '
        else:
            print ('error!!!!!!!!!!!!!!')
            print ()

        if e.title not in entries:
            entries.setdefault(e.title,[summary,d.feed.title])

f=open('summary.txt','w',encoding='utf-8')
i=0
for entry in entries:
    if i<100:
        out.write(entry+'\t'+entries[entry][1].split(' ')[2]+'\n')
        f.write(entries[entry][0]+'\n')
        i=i+1
    else: break
out.close()
f.close()
```

Results:

Books.xml
Dance.xml
Movies.xml
Music.xml
entries.txt
summary.txt

2 50/50 Prediction

Train the Fisher classifier on the first 50 entries, then use the classifier to guess the classification of the next 50 entries.

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

Algorithm:

1. Open "entries.txt" and "summary.txt" to read the title, summary and category of each entry.
2. Use the script from "Programming Collective Intelligence" to train the Fisher classifier on the first 50 entries, then use the classifier to guess the classification of the next 50 entries and save the prediction to "Q2.txt".
3. Compute the precision, recall, and F-measure of each category as well as the "macro-averaged" of all categories and print the result.

Source code:

Listing 2: The content of Q2.py

```
import re
import math

def readfile(filename):
    f = open(filename, 'r')
    lines=f.readlines()
    f.close()
    #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 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):
```

```
    self.fc.setdefault(f,{})
    self.fc[f].setdefault(cat,0)
    self.fc[f][cat]+=1

# Increase the count of a category
def incc(self,cat):
    self.cc.setdefault(cat,0)
    self.cc[cat]+=1

# The number of times a feature has appeared in a category
def fcount(self,f,cat):
    if f in self.fc and cat in self.fc[f]:
        return float(self.fc[f][cat])
    return 0.0

# The number of items in a category
def catcount(self,cat):
    if cat in self.cc:
        return float(self.cc[cat])
    return 0

# The list of all categories
def categories(self):
    return self.cc.keys( )

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)

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 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)

    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


cl=fisherclassifier(getwords)

f=open('summary.txt','r',encoding='utf-8')
summary=f.readlines()
f.close()
f=open('entries.txt','r',encoding='utf-8')
lines=f.readlines()
f.close()

categories={}
for i in range(50):
    cat=lines[i+1].strip().split('\t')[1]
    cl.train(summary[i].strip(),cat)
    if cat not in categories:
        categories.setdefault(cat,[0,0,0])

result=[]
f=open('Q2.txt','w',encoding='utf-8')
f.write('title\tactual\tpredicted\n')
for i in range(50,100):
    line=lines[i+1].strip().split('\t')
    result.append([line[1],cl.classify(summary[i].strip())])
    f.write(line[0]+'\t'+result[i-50][0]+'\t'+result[i-50][1]+'\n')
f.close()
```

```
for i in range(50):
    if result[i][0]==result[i][1]:
        categories[result[i][0]][0]+=1
    else:
        categories[result[i][0]][1]+=1
        categories[result[i][1]][2]+=1

print ('Category\tPrecision\tRecall\tF-Measure')
TP=0
FP=0
FN=0
for cat in categories:
    TP+=categories[cat][0]
    FN+=categories[cat][1]
    FP+=categories[cat][2]
    precision=categories[cat][0]/(categories[cat][0]+categories[cat][2])
    recall=categories[cat][0]/(categories[cat][0]+categories[cat][1])
    print (cat+'\t{}'.format(precision)+'\t{}'.format(recall)
           +'\t{}'.format(2*precision*recall/(precision+recall)))

TP=TP/len(categories)
FP=FP/len(categories)
FN=FN/len(categories)
precision=TP/(TP+FP)
recall=TP/(TP+FN)
print ('Average\t{}'.format(precision)+'\t{}'.format(recall)
       +'\t{}'.format(2*precision*recall/(precision+recall)))
```

Results:

Q2.txt

Table 1: 50/50 Prediction Performance Assessment

Category	Precision	Recall	F-measure
Movies	0.600	0.400	0.480
Music	0.429	0.400	0.414
Dance	0.167	0.600	0.261
Books	0.375	0.200	0.261
Average	0.360	0.360	0.360

3 90/10 Prediction

Repeat question #2, but use the first 90 entries to train your classifier and the last 10 entries for testing.

Algorithm:

1. Open "entries.txt" and "summary.txt" to read the title, summary and category of each entry.
2. Use the script from "Programming Collective Intelligence" to train the Fisher classifier on the first 90 entries, then use the classifier to guess the classification of the next 10 entries and save the prediction to "Q3.txt".
3. Compute the precision, recall, and F-measure of each category as well as the "macro-averaged" of all categories and print the result.(let the recall be 1 if TP=FN=0)

Source code:

Listing 3: The content of Q3.py

```
import re
import math

def readfile(filename):
    f = open(filename, 'r')
    lines=f.readlines()
    f.close()
    #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 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):
        self.fc.setdefault(f,{})
        self.fc[f].setdefault(cat,0)
        self.fc[f][cat]+=1

    # Increase the count of a category
    def incc(self,cat):
        self.cc.setdefault(cat,0)
        self.cc[cat]+=1

    # The number of times a feature has appeared in a category
    def fcount(self,f,cat):
        if f in self.fc and cat in self.fc[f]:
            return float(self.fc[f][cat])
        return 0.0

    # The number of items in a category
    def catcount(self,cat):
        if cat in self.cc:
            return float(self.cc[cat])
        return 0

    # The list of all categories
    def categories(self):
        return self.cc.keys( )

    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)

    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 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)
```

```
    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


cl=fisherclassifier(getwords)

f=open('summary.txt','r',encoding='utf-8')
summary=f.readlines()
f.close()
f=open('entries.txt','r',encoding='utf-8')
lines=f.readlines()
f.close()

categories={}
for i in range(90):
    cat=lines[i+1].strip().split('\t')[1]
    cl.train(summary[i].strip(),cat)
    if cat not in categories:
```

```
        categories.setdefault(cat,[0,0,0])

result=[]
f=open('Q3.txt','w',encoding='utf-8')
f.write('title\tactual\tpredicted\n')
for i in range(90,100):
    line=lines[i+1].strip().split('\t')
    result.append([line[1],cl.classify(summary[i].strip())])
    f.write(line[0]+'\t'+result[i-90][0]+'\t'+result[i-90][1]+'\n')
f.close()

for i in range(10):
    if result[i][0]==result[i][1]:
        categories[result[i][0]][0]+=1
    else:
        categories[result[i][0]][1]+=1
        categories[result[i][1]][2]+=1

print ('Category\tPrecision\tRecall\tF-Measure')
TP=0
FP=0
FN=0
for cat in categories:
    TP+=categories[cat][0]
    FN+=categories[cat][1]
    FP+=categories[cat][2]
    if categories[cat][0]+categories[cat][2]==0:
        precision=1
    else:
        precision=categories[cat][0]/(categories[cat][0]+categories[cat][2])
    if categories[cat][0]+categories[cat][1]==0:
        recall=1
    else:
        recall=categories[cat][0]/(categories[cat][0]+categories[cat][1])
    print (cat+'\t{}'.format(precision)+'\t{}'.format(recall)
           +'\t{}'.format(2*precision*recall/(precision+recall)))

TP=TP/len(categories)
FP=FP/len(categories)
FN=FN/len(categories)
precision=TP/(TP+FP)
recall=TP/(TP+FN)
print ('Average\t{}'.format(precision)+'\t{}'.format(recall)
       +'\t{}'.format(2*precision*recall/(precision+recall)))
```

Results:

Q3.txt

Table 2: 90/10 Prediction Performance Assessment

Category	Precision	Recall	F-measure
Dance	0.00	1.00	0.00
Books	1.00	0.75	0.86
Movies	1.00	0.33	0.50
Music	0.50	0.67	0.57
Average	0.60	0.60	0.60