Web Science cs532: Assignment #10

Due on Saturday, April 30, 2016

Dr.Michael.L.Nelson 4:20pm

Zetan Li

Contents

Problem 1

Using the data from A8:

- Consider each row in the blog-term matrix as a 500 dimension vector, corresponding to a blog.

- From chapter 8, replace numpredict.euclidean() with cosine as the distance metric. In other words, you'll be computing the cosine between vectors of 500 dimensions.

- Use knnestimate() to compute the nearest neighbors for both:

`http://f-measure.blogspot.com/`
`http://ws-dl.blogspot.com/`

for k=1,2,5,10,20.

SOLUTION

We copy the data file from assignment 8 to compute the knn of f-measure and ws_blog.
However, we have to implement the cosine ourselves so, we modified the interface of getdistance and knnestimate function to allow customized distance function.
Another thing is, this cosine = 1 means two vector are overlap each other. So we have to sort the result in descent order. Thus in the code, we have to set the reverse to true when calling sort() in python.

Listing 1: Modified version of numpredict.py (New function cosine() and modified getdistance())

```
from random import random,randint
import math

def wineprice(rating,age):
  peak_age=rating-50

  # Calculate price based on rating
  price=rating/2
  if age>peak_age:
    # Past its peak, goes bad in 10 years
    price=price*(5-(age-peak_age)/2)
  else:
    # Increases to 5x original value as it
    # approaches its peak
    price=price*(5*((age+1)/peak_age))
  if price<0: price=0
  return price


def wineset1():
  rows=[]
  for i in range(300):
    # Create a random age and rating
    rating=random()*50+50
    age=random()*50

    # Get reference price
    price=wineprice(rating,age)
```

```
    # Add some noise
    price*=(random()*0.2+0.9)

    # Add to the dataset
    rows.append({'input':(rating,age),
                 'result':price})
  return rows

def cosine(v1,v2):
  sumab=sum([a*b for a,b in zip(v1,v2)])
  sumasqr=math.sqrt(sum([a*a for a in v1]))
  sumbsqr=math.sqrt(sum([b*b for b in v2]))
  return sumab/(sumasqr*sumbsqr)

def euclidean(v1,v2):
  d=0.0
  for i in range(len(v1)):
    d+=(v1[i]-v2[i])**2
  return math.sqrt(d)


def getdistances(data,vec1,distance=euclidean):
  distancelist=[]

  # Loop over every item in the dataset
  for i in range(len(data)):
    vec2=data[i]

    # Add the distance and the index
    distancelist.append((distance(vec1,vec2),i))

  # Sort by distance
  distancelist.sort(reverse=True)
  return distancelist

def knnestimate(data,vec1,k=5,distance=euclidean):
  # Get sorted distances
  dlist=getdistances(data,vec1,distance)
  avg=0.0

  # Take the average of the top k results
  # for i in range(k):
  #   idx=dlist[i][1]
  #   avg+=data[idx]['result']
  # avg=avg/k
  # return avg
  return dlist

def inverseweight(dist,num=1.0,const=0.1):
  return num/(dist+const)

def subtractweight(dist,const=1.0):
```

```
  if dist>const:
    return 0
  else:
    return const-dist

def gaussian(dist,sigma=5.0):
  return math.e**(-dist**2/(2*sigma**2))

def weightedknn(data,vec1,k=5,weightf=gaussian):
  # Get distances
  dlist=getdistances(data,vec1)
  avg=0.0
  totalweight=0.0

  # Get weighted average
  for i in range(k):
    dist=dlist[i][0]
    idx=dlist[i][1]
    weight=weightf(dist)
    avg+=weight*data[idx]['result']
    totalweight+=weight
  if totalweight==0: return 0
  avg=avg/totalweight
  return avg

def dividedata(data,test=0.05):
  trainset=[]
  testset=[]
  for row in data:
    if random()<test:
      testset.append(row)
    else:
      trainset.append(row)
  return trainset,testset

def testalgorithm(algf,trainset,testset):
  error=0.0
  for row in testset:
    guess=algf(trainset,row['input'])
    error+=(row['result']-guess)**2
    #print row['result'],guess
  #print error/len(testset)
  return error/len(testset)

def crossvalidate(algf,data,trials=100,test=0.1):
  error=0.0
  for i in range(trials):
    trainset,testset=dividedata(data,test)
    error+=testalgorithm(algf,trainset,testset)
  return error/trials

def wineset2():
  rows=[]
```

```
  for i in range(300):
    rating=random()*50+50
    age=random()*50
    aisle=float(randint(1,20))
    bottlesize=[375.0,750.0,1500.0][randint(0,2)]
    price=wineprice(rating,age)
    price*=(bottlesize/750)
    price*=(random()*0.2+0.9)
    rows.append({'input':(rating,age,aisle,bottlesize),
                 'result':price})
  return rows

def rescale(data,scale):
  scaleddata=[]
  for row in data:
    scaled=[scale[i]*row['input'][i] for i in range(len(scale))]
    scaleddata.append({'input':scaled,'result':row['result']})
  return scaleddata

def createcostfunction(algf,data):
  def costf(scale):
    sdata=rescale(data,scale)
    return crossvalidate(algf,sdata,trials=20)
  return costf

weightdomain=[(0,10)]*4

def wineset3():
  rows=wineset1()
  for row in rows:
    if random()<0.5:
      # Wine was bought at a discount store
      row['result']*=0.6
  return rows

def probguess(data,vec1,low,high,k=5,weightf=gaussian):
  dlist=getdistances(data,vec1)
  nweight=0.0
  tweight=0.0

  for i in range(k):
    dist=dlist[i][0]
    idx=dlist[i][1]
    weight=weightf(dist)
    v=data[idx]['result']

    # Is this point in the range?
    if v>=low and v<=high:
      nweight+=weight
    tweight+=weight
  if tweight==0: return 0

  # The probability is the weights in the range
```

```
  # divided by all the weights
  return nweight/tweight

from pylab import *

def cumulativegraph(data,vec1,high,k=5,weightf=gaussian):
  t1=arange(0.0,high,0.1)
  cprob=array([probguess(data,vec1,0,v,k,weightf) for v in t1])
  plot(t1,cprob)
  show()


def probabilitygraph(data,vec1,high,k=5,weightf=gaussian,ss=5.0):
  # Make a range for the prices
  t1=arange(0.0,high,0.1)

  # Get the probabilities for the entire range
  probs=[probguess(data,vec1,v,v+0.1,k,weightf) for v in t1]

  # Smooth them by adding the gaussian of the nearby probabilites
  smoothed=[]
  for i in range(len(probs)):
    sv=0.0
    for j in range(0,len(probs)):
      dist=abs(i-j)*0.1
      weight=gaussian(dist,sigma=ss)
      sv+=weight*probs[j]
    smoothed.append(sv)
  smoothed=array(smoothed)

  plot(t1,smoothed)
  show()
```

Listing 2: Code to compute k nearest neighbors of given blogs

```
#!/usr/bin/python
from numpredict import *
fmeasure='F-Measure'
wlblog='Web Science and Digital Libraries Research Group'
vectors={}
vectorfm=[]
vectorwb=[]
datafile=open('blogdata.txt')
strline=datafile.readlines()
header=True
for line in strline:
     if header:
          #skip header
          header=False
          continue
     tuples=line.strip().split('\t')
     if tuples[0] == fmeasure:
          for i in range(1,len(tuples)):
               vectorfm.append(float(tuples[i]))
```

```
     elif tuples[0]==wlblog:
          for i in range(1,len(tuples)):
               vectorwb.append(float(tuples[i]))
     else:
          vectors[tuples[0]]=[]
          for i in range(1,len(tuples)):
               vectors[tuples[0]].append(float(tuples[i]))
datafile.close()

nn=knnestimate(vectors.values(),vectorfm,distance=cosine)
print ('----K nearest neighbors of F-Measure------------')
for k in [1,2,5,10,20]:
     print('k = %d'%k)
     for j in range(k):
          print('%s\t%.6f'%(vectors.keys()[nn[j][1]],nn[j][0]))
     print('')

nn=knnestimate(vectors.values(),vectorwb,distance=cosine)
print ('----K nearest neighbors of Web Science and Digital Libraries Research
    Group------------')
for k in [1,2,5,10,20]:
     print('k = %d'%k)
     for j in range(k):
          print('%s\t%.6f'%(vectors.keys()[nn[j][1]],nn[j][0]))
     print('')
```

Below is the result.
The format is blog name / cosine value between two blogs.

Listing 3: K nearest neighbor of two given blogs

```
----K nearest neighbors of F-Measure------------
k = 1
The Jeopardy of Contentment   0.539114

k = 2
The Jeopardy of Contentment   0.539114
KiDCHAIR  0.533576

k = 5
The Jeopardy of Contentment   0.539114
KiDCHAIR  0.533576
The Listening Ear   0.520974
Pithy Title Here    0.513004
The Girl at the Rock Show     0.504671

k = 10
The Jeopardy of Contentment   0.539114
KiDCHAIR  0.533576
The Listening Ear   0.520974
Pithy Title Here    0.513004
The Girl at the Rock Show     0.504671
In the Frame Film Reviews     0.498802
DaveCromwell Writes 0.496213
```

```
The Power of Independent Trucking  0.490837
The World's First Internet Baby    0.483861
Tremble Under Boom Lights     0.480495

k = 20
The Jeopardy of Contentment   0.539114
KiDCHAIR  0.533576
The Listening Ear   0.520974
Pithy Title Here    0.513004
The Girl at the Rock Show     0.504671
In the Frame Film Reviews     0.498802
DaveCromwell Writes 0.496213
The Power of Independent Trucking  0.490837
The World's First Internet Baby    0.483861
Tremble Under Boom Lights     0.480495
My Name Is Blue Canary   0.478492
Steel City Rust     0.465259
Doginasweater's Music Reviews (And Other Horseshit)    0.463919
MTJR RANTS & RAVES ON MUSIC   0.463825
I/LOVE/TOTAL/DESTRUCTION 0.455910
The Ideal Copy 0.455147
.    0.450656
Did Not Chart  0.441839
Encore    0.436344
But She's Not Stupid     0.436058

----K nearest neighbors of Web Science and Digital Libraries Research Group
    ------------
k = 1
The Ideal Copy 0.507343

k = 2
The Ideal Copy 0.507343
Samtastic! Review   0.474969

k = 5
The Ideal Copy 0.507343
Samtastic! Review   0.474969
from a voice plantation  0.445301
Karl Drinkwater     0.437251
Eli Jace | The Mind Is A Terrible Thing To Paste  0.396151

k = 10
The Ideal Copy 0.507343
Samtastic! Review   0.474969
from a voice plantation  0.445301
Karl Drinkwater     0.437251
Eli Jace | The Mind Is A Terrible Thing To Paste  0.396151
Tremble Under Boom Lights     0.393766
Brian's Music Blog!!!    0.393500
Pithy Title Here    0.372455
DaveCromwell Writes 0.367888
Kid F     0.364489
```

```
k = 20
The Ideal Copy 0.507343
Samtastic! Review   0.474969
from a voice plantation  0.445301
Karl Drinkwater     0.437251
Eli Jace | The Mind Is A Terrible Thing To Paste  0.396151
Tremble Under Boom Lights     0.393766
Brian's Music Blog!!!    0.393500
Pithy Title Here    0.372455
DaveCromwell Writes 0.367888
Kid F     0.364489
My Name Is Blue Canary   0.343915
I/LOVE/TOTAL/DESTRUCTION 0.341941
But She's Not Stupid     0.333981
The Power of Independent Trucking  0.326787
Sonology  0.324202
For the Other Things     0.319859
A Wife's Tale  0.311471
Morgan's Blog  0.311178
MPC  0.310446
MTJR RANTS & RAVES ON MUSIC   0.307793
```

Problem 2

Rerun A9, Q2 but this time using LIBSVM. If you have n categories, you'll have to run it n times. For example, if you're classifying music and have the categories:

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

you'll have to classify things as:

metal / not-metal
electronic / not-electronic
ambient / not-ambient

etc.

Use the 500 term vectors describing each blog as the features, and your mannally assigned classifications as the true values. Use 10-fold cross-validation (as per slide 46, which shows 4-fold cross-validation) and report the percentage correct for each of your categories.

SOLUTION

First we have to run the word counting script in assignment 8, but this time we modified the word counting on every entries instead of feeds.
Second, instead of use libsvm shown in the slide, we use the sklearn SVC, which implement based on libsvm but with more friendly interface, for classification.

Listing 4: Script to get word count data from one feed on each entries

```
#!/usr/bin/python
```

```
import feedparser
import re

def get_pure_text(text):
  t=re.compile(r'<[^>]+>')
  return t.sub('',text)
# Returns title and dictionary of word counts for an RSS feed
def getwordcounts(title,summary):
  wc={}

  # Extract a list of words
  words=getwords(title+' '+summary)
  for word in words:
    wc.setdefault(word.strip(),0)
    wc[word]+=1
  return title,wc

def getwords(html):
  # Remove all the HTML tags
  txt=re.compile(r'<[^>]+>').sub('',html)

  # Split words by all non-alpha characters
  words=re.compile(r'[^A-Z^a-z]+').split(txt)

  # Convert to lowercase
  return [word.lower() for word in words if word!='']

#-------script entry--------
apcount={}
wordcounts={}
f=feedparser.parse('rawRSS.txt')
counter=0
for entry in f['entries']:
  title = get_pure_text(entry['title'].encode('utf-8'))
  summary = get_pure_text(entry['summary'].encode('utf-8'))

  title,wc=getwordcounts(title,summary)
  wordcounts[title]=wc
  for word,count in wc.items():
    apcount.setdefault(word,0)
    if count>1:
      apcount[word]+=1
  counter+=1
  if counter>=100:
    break

wordlist=[]
for w,bc in apcount.items():
  frac=float(bc)/100
  #if frac>0.01 and frac<0.5:
  wordlist.append(w)
  if len(wordlist)>=500 :
    break
```

```
55
out=file('feedData.txt','w')
out.write('Blog')
for word in wordlist: out.write('\t%s' % word)
out.write('\n')
60 for blog,wc in wordcounts.items():
  #print blog
  try:
    out.write(blog)
  except:
65   out.write(str(blog.encode('utf-8')))
  for word in wordlist:
    if word in wc: out.write('\t%d' % wc[word])
    else: out.write('\t0')
  out.write('\n')
```

Since we discovered the vocabulary to describe a game is so limited to a small scale, we counted all the words that appeared in summary and title, including stop word. Just to make up 500 different words.

Listing 5: svm classifacation based on scikit-learn

```
#!/usr/bin/python
from sklearn import svm
from sklearn import cross_validation
import numpy as np
5 def process_category(category,data):
    svc=svm.SVC(C=10)
    X=[]
    Y=[]
    for unit in data:
10      vec=data[unit]['vector']
        X.append(vec)
        if data[unit]['actual']!=category:
            Y.append(-1)
        else:
15          Y.append(1)

    dataX=np.array(X)
    dataY=np.array(Y)
    svc=svm.SVC(C=10)
20  svc.fit(dataX,dataY)
    score=cross_validation.cross_val_score(svc,dataX,dataY,cv=10)
    return score

#-----script entry
25 games={}
#read raw data
header=True
for line in file('feedData.txt'):
    #skip header
30  if header:
        header=False
        continue
```

```
    tuples=line.strip().split('\t')
    gameName=tuples[0]
    games[gameName]={}
    games[gameName]['vector']=[float(wc) for wc in tuples[1:]]
#read category file
for line in file('p2_table.txt'):
    tuples=line.strip().split('\t')
    gname=tuples[0]
    games[gname]['actual']=tuples[2]
#classification
catetories=['fighting','sports','rpg','arpg','racing','platform','action','fps']
for c in catetories:
    performance=process_category(c,games)
    print('Category: %s'%c)
    for score in performance:
        print(score),
    print(score.mean())
```

Table below is the result of percentage correct.

Table 1: Correctness of svm on each category

Category	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Fold 7**	**Fold 8**	**Fold 9**	**Fold 10**	**Mean**
fighting	0.90	0.90	0.90	0.90	0.90	1.00	1.00	1.00	1.00	1.00	1.00
sports	0.90	0.90	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
rpg	0.72	0.81	0.80	0.80	0.80	0.80	0.77	0.77	0.77	0.88	0.88
arpg	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	1.00	1.00
racing	0.90	0.90	0.90	0.90	0.90	0.90	1.00	1.00	1.00	1.00	1.00
platform	0.80	0.90	0.80	0.80	0.80	0.80	0.80	0.70	0.77	0.66	0.66
action	0.80	0.70	0.70	0.70	0.70	0.70	0.70	0.60	0.88	0.77	0.77
fps	0.90	0.90	0.90	1.00	1.00	0.90	1.00	1.00	1.00	1.00	1.00