MIDS-W261-2015-HWK-Week01-Sakhamuri

January 22, 2016

0.1 Name : Vamsi Sakhamuri

0.2 E-mail : vamsi@ischool.berkeley.edu

0.3 Class Name : W261-3

0.4 Week Number : 1

0.5 Date of submission : 01/19/2016

0.5.1 *HW1.0.0. Define big data. Provide an example of a big data problem in your domain of expertise.*

Big data is a broad term for datasets so large that traditional data processing techniques are inadequate. The data is too big (volume) , moves too fast(velocity) and consists of different types (structured and unstructured - variety) for traditional techniques to be useful and therefore new state-of-the-art methods have to be used to process and derive insights from such big data.

I work as an ASIC(Application Specific Integrated Circuit) Design Engineer at a semiconductor firm in the San Francisco Bay Area. I design ASIC's for Solid State Drives (SSDs). Owing to several benefits over hard-disks, solid state drives are nowadays ubiquitous in laptops,desktops, data-centers etc. Solid State Drive is a storage device and under the hood consists of a ASIC controller chip and several NAND Flash memory chips. These NAND Flash memory chips are what store the bits. At a more granular level , there is a special type of transistor (called the floating-gate transistor) which stores a bit. There are several billion of these floating gate type transistors in a solid state drive system. Each Solid state drive can easily have a density of 1 TB these days (approximately 4096 billion floating gate transistors). Each of these transistors exhibit slightly different physical characteristics and the challenge is to analyze the reliability of each of these transistors before shipping them to the customer. And on top of it, there are hundreds of thousands of these solid state drives being shipped. So, the amount of data to be analyzed can get "big".

0.5.2 *HW1.0.1.In 500 words (English or pseudo code or a combination) describe how to estimate the bias, the variance, the irreducible error for a test dataset T when using polynomial regression models of degree 1, 2,3, 4,5 are considered. How would you select a model?*

Introduction:

Bias and variance are closely related to overfitting and underfitting of data. Our goal should be to select a model which has a low bias as well as a low variance. However, there is a tradeoff between bias and variance and our goal should be to minimize both of them simultaneouly.

Bias: Bias is the difference between the expected value of an estimator and the true value of the function

Bias = E[g(x)] - f(x)

where g(x) is the estimator and f(x) is the true function.

In order to determine bias, we should repeatedly sample the data and then fit our model to all the different samples. Once we fit our model to all the different samples, then we should calculate the mean

predicted value obtained from our model across all these samples ,i.e expected value of the estimator for all the samples. This is E[g(x)].

Then, simply subtracting f(x) from E[g(x)] gives us the bias of our model

Pseudo Code:

Lets assume we will sample our data N times:

1) for (m =1;m<6; m=m+1) (outer loop - choose a polynomial model of degree m)

2) for (i=0;i<N;i=i+1) (inner loop)

```
data from the ith sample

fit our model to the ith sample

store this fit in g(xi)
```

Compute the mean of all the fits, g(x0),g(x1),g(xN) where the data samples go from 0 to N

This gives us E[g(x)].

Bias[m] = E[g(x)] - f(x) // bias of polynomial model of degree m

Return to outer loop

Variance: Variance tells us how much our estimator is expected to vary for different samples of data.

Variance = E[g(x) - E[g(x)]]ˆ2

where g(x) is the estimator

Pseudo Code:

Lets assume we will sample our data N times:

1)for (m =1;m<6; m=m+1) (outer loop - choose a polynomial model of degree m)

2) for (i=0;i<N;i=i+1) (inner loop)

```
data from the ith sample

fit our model to the ith sample

store this fit in g(xi)
```

Compute the mean of all the fits, g(x0),g(x1),g(xN) where the data samples go from 0 to N

This gives us E[g(x)].

Variance[m] = E[g(x) - E[g(x)]]ˆ2 // variance of polynomial model of degree m

Return to outer loop

Irreducible error: Irreducible error is the noise component of our prediction and does not depend on the estimator.

Variance of noise = E[(y - f(x)]ˆ2

where y is the output value from our sample and f(x) is the true function

In general , y = f(x) + e , where e is the noise term

0.5.3 Model Selection:

From the bias and variance pseudocode described above, we have the bias and variance of all the polynomial models of degrees 1,2,3,4 and 5.

We will plot these bias variance values on the y-axis against the model complexity (degree) on the x-axis and choose the model where the biasˆ2 +variance value is the smallest.

0.5.4 HW1.1. Read through the provided control script (pNaiveBayes.sh) and all of its comments. When you are comfortable with their purpose and function, respond to the remaining homework questions below. A simple cell in the notebook with a print statmement with a "done" string will suffice here. (dont forget to include the Question Number and the quesition in the cell as a multiline comment!)

```
In [818]: print("done")
```

```
done
```

0.5.5 HW1.2. Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will determine the number of occurrences of a single, user-specified word. Examine the word "assistance" and report your results.

```
In [819]: %%writefile mapper.py
          #!/usr/bin/python
          import sys
          import re
          count = 0
          WORD_RE = re.compile(r"[\w']+",re.IGNORECASE)
          filename = sys.argv[1]
          findword = sys.argv[2]

          with open (filename, "r") as myfile:
              mylist = []
              for line in myfile:
                  mylist = line.split()
                  for each_word in mylist:
                      hit = re.search(findword,each_word,re.IGNORECASE)
                      if(hit):
                          count = count + 1

          print count
```

```
Overwriting mapper.py
```

```
In [820]: !chmod a+x mapper.py
```

```
In [821]: %%writefile reducer.py
          #!/usr/bin/python
          import sys
          sum = 0
          filename = sys.argv[1:]

          for f in filename:
              myList = []
              with open (f,"r") as fn:
                  for line in fn:
                      myList = line.split()
                      for line in myList:
                          sum = sum + int(line)
          print sum
```

```
Overwriting reducer.py
```

```
In [822]: !chmod a+x reducer.py
```

In [823]:
```
!chmod a+x pNaiveBayes.sh
```

In [824]:
```
!./pNaiveBayes.sh 4 "assistance"
```

"assistance" occurs 10 times in the email corpus

0.5.6 HW1.3. Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by a single, user-specified word using the multinomial Naive Bayes Formulation. Examine the word "assistance" and report your results

In [825]:
```
%%writefile mapper.py
#!/usr/bin/python
import sys
import re
count = 0
WORD_RE = re.compile(r"[\w']+",re.IGNORECASE)
filename = sys.argv[1]
findword = sys.argv[2]

with open (filename, "r") as myfile:
    mylist = []
    master_list = []
    for line in myfile:
        count_hit = 0
        count_all = 0
        doc_list = []
        email = re.split(r'\t+',line)
        if len(email)==4:
            i=0
            for each_field in email:
                if(i==0):
                    doc_list.append(each_field)   #doc ID
                elif (i==1):
                    doc_list.append(each_field)   #spam=1 or 0=ham
                elif (i==2):
                    all_words = each_field.split()
                    for w in all_words:
                        hit = re.search(findword,w,re.IGNORECASE)
                        if(hit):
                            count_hit = count_hit+1          #search in the subject of the mes
                    all_words = each_field.split()
                    count_all = count_all + len(all_words)

                elif (i==3):
                    all_words = each_field.split()
                    for w in all_words:
                        hit = re.search(findword,w,re.IGNORECASE)
                        if(hit):
                            count_hit = count_hit+1          #search in the body of the messag
                    count_all = count_all + len(all_words)

                i=i+1
            doc_list.append(count_hit)
            doc_list.append(count_all)
            doc_list.append(len(email))
```

```
                    #Each email is transformed into a list [docID,spam or not,occurences of selected w
                    #                                       total word count in email,number of parts

                master_list.append(doc_list)
        for doc in master_list:         #master_list is a list of lists of doc_lists
            print doc
```

```
Overwriting mapper.py
```

In [826]:
```
!chmod a+x mapper.py
```

In [827]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import re
from math import log

filename = sys.argv[1:]

total_email_count = 0
spam_email_count = 0
spam_total_word_count = 0
spam_user_word_count = 0
ham_email_count = 0
ham_total_word_count = 0
ham_user_word_count = 0

#Step 1 - Figure out the priors and conditionals from all the mapper outputs
for f in filename:
    with open (f,"r") as fn:
        for line in fn:
            r0=re.sub('\[','',line)
            r1=re.sub('\]','',r0)
            r2=re.sub('\"','',r1)
            r3=re.sub('\'','',r2)
            my_list = r3.split(',')
            if(int(my_list[1])==1):
                spam_email_count = spam_email_count+1
                spam_total_word_count = spam_total_word_count + int(my_list[3])
                spam_user_word_count = spam_user_word_count + int(my_list[2])
            elif(int(my_list[1])==0):
                ham_email_count = ham_email_count+1
                ham_total_word_count = ham_total_word_count + int(my_list[3])
                ham_user_word_count = ham_user_word_count + int(my_list[2])
            total_email_count = total_email_count+1

spam_prior = float(spam_email_count)/float(total_email_count)
ham_prior = float(ham_email_count)/float(total_email_count)
conditional_word_given_spam = float(spam_user_word_count)/float(spam_total_word_count)
conditional_word_given_ham = float(ham_user_word_count)/float(ham_total_word_count)

tt = 0
#Step 2 - Classification
prediction = []
for f in filename:
```

```
            with open (f,"r") as fn:
                for line in fn:
                    r0=re.sub('\[','',line)
                    r1=re.sub('\]','',r0)
                    r2=re.sub('\"','',r1)
                    r3=re.sub('\'','',r2)
                    my_list = r3.split(',')

                    P_spam_given_word = spam_prior + ((conditional_word_given_spam)**int(my_list[2]))
                    P_ham_given_word = ham_prior + ((conditional_word_given_ham)**int(my_list[2]))  #

                    if(P_spam_given_word > P_ham_given_word):
                        prediction.append([1,int(my_list[1]),P_spam_given_word,P_ham_given_word,int(my
                        tt = tt+int(my_list[2])
                    else:
                        prediction.append([0,int(my_list[1]),P_spam_given_word,P_ham_given_word,int(my
                        tt = tt+int(my_list[2])
        good = 0
        total = 0
        for p in prediction:
            if(p[0]==p[1]):
                good = good+1
            total = total+1

        accuracy = 100*float(good)/float(total)
        print accuracy
        print conditional_word_given_spam
        print conditional_word_given_ham
```

```
Overwriting reducer.py
```

```
In [828]: !chmod a+x reducer.py
```

```
In [829]: !chmod a+x pNaiveBayes.sh
```

```
In [830]: !./pNaiveBayes.sh 4 "assistance"
```

Accuracy : 56.0%
P(word|SPAM) = 0.000421585160202
P(word|HAM) = 0.000143513203215

0.5.7 HW1.4. Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by a list of one or more user-specified words. Examine the words "assistance", "valium", and "enlargementWithATypo" and report your results

```
In [831]: %%writefile mapper.py
          #!/usr/bin/python
          import sys
          import re
          count = 0
          WORD_RE = re.compile(r"[\w']+",re.IGNORECASE)
          filename = sys.argv[1]
          findword = sys.argv[2]

          with open (filename, "r") as myfile:
              mylist = []
```

```
master_list = []
for line in myfile:
    count_hit_w0 =0
    count_hit_w1 = 0
    count_hit_w2 = 0
    count_all = 0
    doc_list = []
    email = re.split(r'\t+',line)
    if len(email)==4:
        i=0
        for each_field in email:
            if (i==1):
                doc_list.append(each_field)   #spam=1 or 0=ham
            elif (i==2):
                all_words = each_field.split()
                for w in all_words:
                    findword_split = findword.split()
                    w_int = 0
                    for user_word in findword_split:
                        hit = re.search(user_word,w,re.IGNORECASE)
                        if(hit):
                            if(w_int==0):
                                count_hit_w0 = count_hit_w0 +1
                            elif(w_int==1):
                                count_hit_w1 = count_hit_w1 +1
                            elif(w_int==2):
                                count_hit_w2 = count_hit_w2 +1
                        w_int = w_int+1
                count_all = count_all + len(all_words)
            elif (i==3):
                all_words = each_field.split()
                for w in all_words:
                    findword_split = findword.split()
                    w_int = 0
                    for user_word in findword_split:
                        hit = re.search(user_word,w,re.IGNORECASE)
                        if(hit):
                            if(w_int==0):
                                count_hit_w0 = count_hit_w0 +1
                            elif(w_int==1):
                                count_hit_w1 = count_hit_w1 +1
                            elif(w_int==2):
                                count_hit_w2 = count_hit_w2 +1
                        w_int = w_int+1
                count_all = count_all + len(all_words)
            i=i+1

        doc_list.append(count_hit_w0)
        doc_list.append(count_hit_w1)
        doc_list.append(count_hit_w2)

        doc_list.append(count_all)
        #Each email is transformed into a list [spam or not,occurences of selected words,
        #                                       total word count in email]
```

```
            master_list.append(doc_list)
        for doc in master_list:         #master_list is a list of lists of doc_lists
            print doc
```

```
Overwriting mapper.py
```

In [832]:
```
#!chmod a+x mapper.py
```

In [833]:
```
%%writefile reducer.py
#!/usr/bin/python
import sys
import re
from math import log

filename = sys.argv[1:]

total_email_count = 0

spam_email_count = 0
spam_total_word_count = 0
spam_user_word_count_w0 = 0
spam_user_word_count_w1 = 0
spam_user_word_count_w2 = 0

ham_email_count = 0
ham_total_word_count = 0
ham_user_word_count_w0 = 0
ham_user_word_count_w1 = 0
ham_user_word_count_w2 = 0

#Step 1 - Figure out the priors and conditionals from all the mapper outputs
for f in filename:
    with open (f,"r") as fn:
        for line in fn:      #Each line from the mapper is [spam or not, count_w0,count_w1,cou
            r0=re.sub('\[','',line)
            r1=re.sub('\]','',r0)
            r2=re.sub('\"','',r1)
            r3=re.sub('\'','',r2)
            my_list = r3.split(',')
            if(int(my_list[0])==1):
                spam_email_count = spam_email_count+1
                spam_total_word_count = spam_total_word_count + int(my_list[4])
                spam_user_word_count_w0 = spam_user_word_count_w0 + int(my_list[1])
                spam_user_word_count_w1 = spam_user_word_count_w1 + int(my_list[2])
                spam_user_word_count_w2 = spam_user_word_count_w2 + int(my_list[3])
            elif(int(my_list[0])==0):
                ham_email_count = ham_email_count+1
                ham_total_word_count = ham_total_word_count + int(my_list[4])
                ham_user_word_count_w0 = ham_user_word_count_w0 + int(my_list[1])
                ham_user_word_count_w1 = ham_user_word_count_w1 + int(my_list[2])
                ham_user_word_count_w2 = ham_user_word_count_w2 + int(my_list[3])

            total_email_count = total_email_count+1
```

```
spam_prior = float(spam_email_count)/float(total_email_count)
ham_prior = float(ham_email_count)/float(total_email_count)
conditional_word_given_spam_w0 = float(spam_user_word_count_w0)/float(spam_total_word_count)
conditional_word_given_ham_w0 = float(ham_user_word_count_w0)/float(ham_total_word_count)
conditional_word_given_spam_w1 = float(spam_user_word_count_w1)/float(spam_total_word_count)
conditional_word_given_ham_w1 = float(ham_user_word_count_w1)/float(ham_total_word_count)
conditional_word_given_spam_w2 = float(spam_user_word_count_w2)/float(spam_total_word_count)
conditional_word_given_ham_w2 = float(ham_user_word_count_w2)/float(ham_total_word_count)


#Step 2 - Classification
prediction = []
for f in filename:
    with open (f,"r") as fn:
        for line in fn:
            r0=re.sub('\[','',line)
            r1=re.sub('\]','',r0)
            r2=re.sub('\"','',r1)
            r3=re.sub('\'','',r2)
            my_list = r3.split(',')

            c0s = ((conditional_word_given_spam_w0)**int(my_list[1]))
            c1s = ((conditional_word_given_spam_w1)**int(my_list[2]))
            c2s = ((conditional_word_given_spam_w2)**int(my_list[3]))

            c0h = ((conditional_word_given_ham_w0)**int(my_list[1]))
            c1h = ((conditional_word_given_ham_w1)**int(my_list[2]))
            c2h = ((conditional_word_given_ham_w2)**int(my_list[3]))

            P_spam_given_word = (spam_prior) + c0s*c1s*c2s
            P_ham_given_word = (ham_prior) + c0h*c1h*c2h

            if(P_spam_given_word > P_ham_given_word):
                prediction.append([1,int(my_list[0]),P_spam_given_word,P_ham_given_word])
            else:
                prediction.append([0,int(my_list[0]),P_spam_given_word,P_ham_given_word])

good = 0
total = 0
for p in prediction:
    if((p[0])==int(p[1])):
        good = good+1
    total = total+1

accuracy = 100*float(good)/float(total)
print accuracy
print conditional_word_given_spam_w0
print conditional_word_given_ham_w0
print conditional_word_given_spam_w1
print conditional_word_given_ham_w1
print conditional_word_given_spam_w2
print conditional_word_given_ham_w2
```

```
Overwriting reducer.py
```

```
In [834]: !chmod a+x reducer.py

In [835]: !chmod a+x pNaiveBayes.sh

In [836]: !./pNaiveBayes.sh 4 "assistance valium enlargementWithATypo"
```

Accuracy - 56.0%
P(Word0|SPAM) = 0.000421585160202
P(Word0|HAM) = 0.000143513203215
P(Word1|SPAM) = 0.000158094435076
P(Word1|HAM) = 0.0
P(Word2|SPAM) = 0.0
P(Word2|HAM) = 0.0