DATSCIW261 ASSIGNMENT #1

Jing Xu

jaling@gmail.com

W261-3

DATSCIW261 Assignment #1

1/15/16

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

Big data is an umbrella description for existing datasets or sources of new data that haven't been tapped previously that contain a volume and/or complexity of data that can't be feasibly processed using traditional data-processing applications. Previously unquantifiable and uncaptured data, for example in the form of bioinformatic data or user behavior data, are now being generated and logged at a volume and complexity that exceed the hardware capabilities of single machines. These datasets require some form of parallel processing in order to achieve a throughput acceptable for working timelines in industry. Big data For example, in the legal industry, there are projects using natural language processing to automate the classification of all legal and court documents produced daily by every local, state, and federal court in the U.S. These documents amount to gigabytes of data every week and petabytes every year. The current existing corpus of documents are in the petabytes. This has become a big data problem because companies in the legal tech space are looking to build legal research and document generation products that actively query these document databases and run NLP algorithms too complex for a single machine to process. A parallel processing infrastructure is needed to reasonably access and analyze this volume of legal document data.

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?

The Expected prediction error of a regression model can be broken down as Variance + Bias^2 + Irreducible Error (Noise^2)

Bias - The bias of the regression model is calculated using the formula (h - y), where h is the average prediction model fit over all the datasets that can be sampled from the complete population that dataset T is sampled from, and y is the true function that describes the population. In this case, we will need to sample multiple datasets from the complete population to determine h. As the degree of the polynomial increases, the model becomes better fit to the data, which decreases the value of h and decreases the bias of the model.

Variance - The variance of the regression model is calculated using the formula above, where k is each datapoint in T, yk is the model fit over T in question, and K is the total number of parameters in the model. Variance is a measure of the difference between the model dependent on T with the model estimated over all datasets drawn from the complete population. As the degree of the polynomial increases, the model yk becomes better fit to the data but the estimates become farther from the average model h, leading variance to increase.

Irreducible Error - The irreducible error theoretically can't be calculated because we almost never know the true underlying function from which the dataset is generated. It is a noise term that measures the natural difference between the mean estimator fit on all datasets and the true function of the complete population.

I would select a model by incorporating either an AIC or BIC metric. AIC and BIC is a method of adding a penalty term for the number of parameters in a model. I would calculate the AIC or BIC for each polynomial regression model and choose the model where the AIC/BIC gain is balanced out by the increasing complexity of the model. BIC penalizes the model for more parameters more so than AIC, so depending on whether I want the benefits of a more complex model I may use AIC or BIC. Both AIC and BIC are derived from the formula [-2logL + kp], where L is the likelihood function, p is the number of parameters in the model, and k is 2 for AIC and log(n) where n = sample size for BIC. To determine the final best model, I would look at a combination of the AIC/BIC scores along with the bias^2 and variance. I would plot these values for all regression models and observe if there is an optimal lowpoint - if there is a clear model with the lowest scores in all areas, that will most likely be the best model. If the scores are not as clear, I would choose a model based on the type of data, whether I am looking to minimize the bias to get a model closest to the true underlying function or if I'm trying to minimize variance to get a model that would be closest to the estimated best fit model over all possible sample datasets from the population.

In [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.
print "done"
```

```
done
```

In [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 [177]:
```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Jing Xu
## Description: mapper code for HW1.2
import sys
import re
import string

filename = sys.argv[1] #read in first argument as the emails to be parsed
findwords = sys.argv[2] #read in second argument as the word to be counted
print "FINDWORD", findwords

with open (filename, "r") as emails:
    for email in emails.readlines(): #read each line in enronemail file, each corresponding to a single email sent
        words = email.translate(string.maketrans("",""), string.punctuation) #strip all punctuation
        words = words.split() #convert line into list of words
        for word in words:
            print word, 1
```

```
Overwriting mapper.py
```

In [178]:
```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Jing Xu
## Description: reducer code for HW1.2

import sys

findword = None
words = {} #creating unique word list

for filenames in sys.argv[1:]: #open each filename in the countfile
list
    myfile = open('%s'%filenames, "r")
    for line in myfile.readlines(): #read each line in mapper outpu
t
        line = line.split() #split each line into list
        if line[0] == "FINDWORD": findword = line[1]
        else:
            for word in line:
                if word not in words: words[word] = 1
                else: words[word]+=1

print words[findword]
```

```
Overwriting reducer.py
```

In [180]:
```
!chmod a+x reducer.py
!chmod a+x mapper.py
!chmod a+x pNaiveBayes.sh
```

In [183]:
```
!./pNaiveBayes.sh 5 "assistance"
```

There are 10 occurrences of the word "assistance"

In [7]:
```
# HW1.3. Provide a mapper/reducer pair that, when executed by pNaiv
eBayes.sh will classify
# the email messages by a single, user-specified word using the mul
tinomial Naive Bayes Formulation.
# Examine the word “assistance” and report your results.
```

In [204]:
```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Jing Xu
## Description: mapper code for HW1.3
import sys
import re
import string

filename = sys.argv[1]
findwords = sys.argv[2]
emails = open(filename, "r")
print "FINDWORD", findwords

for line in emails.readlines():
    email_id = line.split('\t')[0]
    line = line.translate(string.maketrans("",""), string.punctuati
on) #strip punctuation
    email = re.split(r'\t+', line) #strip words that include any nu
mbers
    if len(email) != 4: #skip over email data formatting errors
        continue
    content = email[2] + email[3] #concatenate subject and body sec
tions into one string
    content = re.sub(r'\w*\d\w*', '', content).strip() #strip all w
ords that include a number as these words are unlikely to be predic
tive
    content = re.sub("\s\s+" , " ", content) #strip all extra white
spaces
    list_content = content.split(' ') #list of each word in line
    if int(email[1]) == 1: #check if the email is spam or not, coun
t instances of word appearing in spam/not-spam emails and total ema
ils
        print "SPAM_COUNT", len(list_content) #emit key of SPAM_COU
NT along with a word count value for later calculation of total SPA
M words
        for word in list_content:
            print email_id, word.lower(), "SPAM", list_content.coun
t(word) #emit email_id key, word, class, and count
    else:
        print "HAM_COUNT", len(list_content) #emit key of HAM_COUNT
along with a word count value for later calculation of total HAM wo
rds
        for word in list_content:
            print email_id, word.lower(), "HAM", list_content.coun
t(word) #emit email_id key, word, class, and count
```

```
Overwriting mapper.py
```

In [205]:
```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Jing Xu
## Description: reducer code for HW1.3

import sys
import re
import string
import ast

spam_emails = 0
ham_emails = 0
total_spam_words = 0
total_ham_words = 0
findword = ''
filename = ''
words = {} #creating unique word list

for filenames in sys.argv[1:]: #open each filename in the countfile
list
    myfile = open('%s'%filenames, "r")
    for line in myfile.readlines(): #read each line in mapper outpu
t
        line = line.split() #split each line into list
        if line[0] == 'SPAM_COUNT':
            spam_emails+=1 #add 1 to spam_emails
            total_spam_words+=int(line[1]) #add spam words in email
to total_spam_words
        elif line[0] == 'HAM_COUNT':
            ham_emails+=1 #add 1 to ham_emails
            total_ham_words+=int(line[1]) #add ham words in email t
o total_spam_words
        elif line[0] == 'FINDWORD': findword = line[1] #store findw
ord in memory
        else:
            word = str(line[1])
            if line[2] == "SPAM": #sort word into SPAM dictionary
                if word not in words: #create new dictionary index
for word if not already existing
                    words[word] = {}
                    words[word]['SPAM'] = int(line[3]) #set count o
f the number of word in spam emails to 1
                else:
                    if 'SPAM' in words[word]: words[word]['SPA
M']+=int(line[3]) #add 1 to number of word in spam emails
                    else: words[word]['SPAM'] = int(line[3]) #if wo
rd exists in dictionary but not the spam count, create spam count f
or the word
            else: #sort word into HAM dictionary
                if word not in words: #create new dictionary index
for word if not already existing
                    words[word] = {}
```

```
                    words[word]['HAM'] = int(line[3]) #set count of
the number of word in ham emails to 1
                else:
                    if 'HAM' in words[word]: words[word]['HAM']+=in
t(line[3]) #add 1 to number of word in ham emails
                    else: words[word]['HAM'] = int(line[3]) #if wor
d exists in dictionary but not the ham count, create spam count for
the word


prior_spam = float(spam_emails)/(float(spam_emails)+float(ham_email
s)) #prior spam = spam emails / total emails
prior_ham = float(ham_emails)/(float(spam_emails)+float(ham_email
s)) #prior ham = ham emails / total emails
spam_probability = float(words[findword]['SPAM'])/float(total_spa
m_words) #spam probability is the number of occurrences of word in
spam emails / total words in spam emails
ham_probability = float(words[findword]['HAM'])/float(total_ham_wor
ds) #ham probability is the number of occurrences of word in ham em
ails / total words in non-spam emails

print "SPAM,", findword, spam_probability
print "HAM,", findword, ham_probability
print "SPAM Prior =", prior_spam
print "HAM Prior =", prior_ham

# for email in all_emails:
#     count_of_word = 0
#     for each in email:
#         if findword == each: #create count of word
#             count_of_word+=1
#             print 'yes'
#     mnb_spam_probability = prior_spam*spam_probability**count_o
f_word #mnb formula for calculating probability of spam given a wor
d
#     mnb_ham_probability = prior_ham*ham_probability**count_of_wor
d #mnb formula for calculating probability of ham given a word
#     if mnb_spam_probability > mnb_ham_probability: predictions.ap
pend(1) #if probability of spam > ham, prediction of 1 indicates sp
am
#     else: predictions.append(0)

# print predictions
```

```
Overwriting reducer.py
```

In [206]:
```
!./pNaiveBayes.sh 5 "assistance"
```

SPAM, assistance 0.000755770013371

HAM, assistance 0.000164758217316

SPAM Prior = 0.438775510204

HAM Prior = 0.561224489796

In [11]:
```
# HW1.4. Provide a mapper/reducer pair that, when executed by pNaiv
eBayes.sh
# will classify the email messages by a list of one or more user-sp
ecified words.
# Examine the words “assistance”, “valium”, and “enlargementWithATy
po” and report your results.
```

In [226]:
```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Jing Xu
## Description: mapper code for HW1.4
import sys
import re
import string

filename = sys.argv[1]
findwords = sys.argv[2]
emails = open(filename, "r")
print "FINDWORDS", findwords

for line in emails.readlines():
    email_id = line.split('\t')[0]
    line = line.translate(string.maketrans("",""), string.punctuati
on) #strip punctuation
    email = re.split(r'\t+', line) #strip words that include any nu
mbers
    if len(email) != 4: #skip over email data formatting errors
        continue
    content = email[2] + email[3] #concatenate subject and body sec
tions into one string
    content = re.sub(r'\w*\d\w*', '', content).strip() #strip all w
ords that include a number as these words are unlikely to be predic
tive
    content = re.sub("\s\s+" , " ", content) #strip all extra white
spaces
    list_content = content.split(' ') #list of each word in line
    if int(email[1]) == 1: #check if the email is spam or not, coun
t instances of word appearing in spam/not-spam emails and total ema
ils
        print "SPAM_COUNT", len(list_content) #emit key of SPAM_COU
NT along with a word count value for later calculation of total SPA
M words
        for word in list_content:
            print email_id, word.lower(), "SPAM", list_content.coun
t(word) #emit email_id key, word, class, and count
    else:
        print "HAM_COUNT", len(list_content) #emit key of HAM_COUNT
along with a word count value for later calculation of total HAM wo
rds
        for word in list_content:
            print email_id, word.lower(), "HAM", list_content.coun
t(word) #emit email_id key, word, class, and count
```

```
Overwriting mapper.py
```

In [247]:
```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Jing Xu
## Description: reducer code for HW1.4

import sys
import re
import string
import ast

spam_emails = 0
ham_emails = 0
total_spam_words = 0
total_ham_words = 0
findwords = ''
filename = ''
words = {} #creating unique word list

for filenames in sys.argv[1:]: #open each filename in the countfile
list
    myfile = open('%s'%filenames, "r")
    for line in myfile.readlines(): #read each line in mapper outpu
t
        line = line.split() #split each line into list
        if line[0] == 'SPAM_COUNT':
            spam_emails+=1 #add 1 to spam_emails
            total_spam_words+=int(line[1]) #add spam words in email
to total_spam_words
        elif line[0] == 'HAM_COUNT':
            ham_emails+=1 #add 1 to ham_emails
            total_ham_words+=int(line[1]) #add ham words in email t
o total_spam_words
        elif line[0] == 'FINDWORDS': findwords = line[1:] #store fi
ndword in memory
        else:
            word = str(line[1])
            if line[2] == "SPAM": #sort word into SPAM dictionary
                if word not in words: #create new dictionary index
for word if not already existing
                    words[word] = {}
                    words[word]['SPAM'] = int(line[3]) #set count o
f the number of word in spam emails to 1
                else:
                    if 'SPAM' in words[word]: words[word]['SPA
M']+=int(line[3]) #add 1 to number of word in spam emails
                    else: words[word]['SPAM'] = int(line[3]) #if wo
rd exists in dictionary but not the spam count, create spam count f
or the word
            else: #sort word into HAM dictionary
                if word not in words: #create new dictionary index
for word if not already existing
                    words[word] = {}
```

```
                    words[word]['HAM'] = int(line[3]) #set count of
the number of word in ham emails to 1
                else:
                    if 'HAM' in words[word]: words[word]['HAM']+=in
t(line[3]) #add 1 to number of word in ham emails
                    else: words[word]['HAM'] = int(line[3]) #if wor
d exists in dictionary but not the ham count, create spam count for
the word

for findword in findwords:
    prior_spam = float(spam_emails)/(float(spam_emails)+float(ham_e
mails)) #prior spam = spam emails / total emails
    prior_ham = float(ham_emails)/(float(spam_emails)+float(ham_ema
ils)) #prior ham = ham emails / total emails
    try: spam_probability = float(words[findword]['SPAM'])/float(to
tal_spam_words) #spam probability is the number of occurrences of w
ord in spam emails / total words in spam emails
    except: spam_probability = 'N/A'
    try: ham_probability = float(words[findword]['HAM'])/float(tota
l_ham_words) #ham probability is the number of occurrences of word
in ham emails / total words in non-spam emails
    except: ham_probability = 'N/A'
    print "Class conditional: SPAM,", findword, spam_probability
    print "Class conditional: HAM,", findword, ham_probability
    print "SPAM Prior =", prior_spam
    print "HAM Prior =", prior_ham
```

```
Overwriting reducer.py
```

In [248]:
```
!./pNaiveBayes.sh 5 "assistance valium enlargementWithATypo"
```

Class conditional: SPAM, assistance 0.000755770013371

Class conditional: HAM, assistance 0.000164758217316

SPAM Prior = 0.438775510204

HAM Prior = 0.561224489796

Class conditional: SPAM, valium 0.000174408464624

Class conditional: HAM, valium N/A

SPAM Prior = 0.438775510204

HAM Prior = 0.561224489796

Class conditional: SPAM, enlargementWithATypo N/A

Class conditional: HAM, enlargementWithATypo N/A

SPAM Prior = 0.438775510204

HAM Prior = 0.561224489796