DATSCIW261 ASSIGNMENT #1

Jing Xu

jaling@gmail.com

W261-3

DATSCIW261 Assignment #1

1/15/16

HW1.0.0. Big data is an umbrella description for datasets that contain a volume and/or complexity of data that can't be feasibly processed using traditional data-processing applications. These datasets require some form of parallel processing in order to achieve a throughput acceptable for working timelines in industry. 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

HW1.0.1.

The total expected error of a regression model can be broken down as Irreducible Error + Bias^2 + Variance

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

Variance - The variance of the regression model is calculated using the formula: variance=E[(g(x)-E[g(x)])^2], where g(x) is the model fit over T. This is to measure 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 g(x) becomes better fit to the data but the estimates become farther from the average model E[g(x)], 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. 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 true population.

In [1]:
```
# 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 "do
ne" string will suffice here.
print "done"
```

```
done
```

In [2]:
```
# HW1.2. Provide a mapper/reducer pair that, when executed by pNaiv
eBayes.sh
# will determine the number of occurrences of a single, user-specif
ied word.
# Examine the word “assistance” and report your results.
```

In [3]:
```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Jing Xu
## Description: mapper code for HW1.2
import sys
import re
import string

count = 0
filename = sys.argv[1]
findwords = sys.argv[2]
with open (filename, "r") as myfile:
    for line in myfile.readlines():
        words = line.translate(string.maketrans("",""), string.punc
tuation) #strip all punctuation
        words = re.split(" ", words.lower()) #convert line into lis
t of words
        for word in words:
            if word.lower() == findwords.lower(): #count each word 
match
                count+=1
print int(count)
```

```
Overwriting mapper.py
```

In [4]:
```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Jing Xu
## Description: reducer code for HW1.2

import sys
total = 0
for filenames in sys.argv[1:]: #open each filename in the countfile
list
    myfile = open('%s'%filenames, "r")
    for line in myfile.readlines():
        total+=int(line) #add each chunk count
print total
```

```
Overwriting reducer.py
```

In [5]:
```
!chmod a+x reducer.py
!chmod a+x mapper.py
!chmod a+x pNaiveBayes.sh
```

In [6]:
```
!./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 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 [8]:
```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Jing Xu
## Description: mapper code for HW1.3
import sys
import re
import string

spam_emails = 0
total_emails = 0
total_spam_words = 0
total_ham_words = 0
spam_count = 0
ham_count = 0
filename = sys.argv[1]
findwords = sys.argv[2]
emails = open(filename, "r")
for line in emails.readlines():
    line = line.translate(string.maketrans("",""), string.punctuati
on) #strip punctuation
    email = re.split(r'\t+', line)
    if len(email) != 4: #skip over email data formatting errors
        continue
    total_emails+=1
    content = email[0] + email[2] + email[3] #concatenate subject a
nd body sections 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
        spam_emails+=1
        for word in list_content:
            if word.lower() == findwords.lower():
                spam_count+=1
            total_spam_words+=1
    else:
        for word in list_content:
            if word.lower() == findwords.lower():
                ham_count+=1
            total_ham_words+=1
    print content

print "spam_emails", spam_emails
print "total_emails", total_emails
print "total_spam_words", total_spam_words
print "total_ham_words", total_ham_words
print "spam_count", spam_count
```

```
print "ham_count", ham_count
print "word", findwords
```

```
Overwriting mapper.py
```

In [9]:
```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Jing Xu
## Description: reducer code for HW1.3

import sys

spam_emails = 0
total_emails = 0
total_spam_words = 0
total_ham_words = 0
spam_count = 0
ham_count = 0
word = ''
all_emails = []
for filenames in sys.argv[1:]: #open each filename in the countfile
list
    myfile = open('%s'%filenames, "r")
    for line in myfile.readlines():
        line = line.split()
        #aggregate counts for all variables of interest
        if line[0] == "spam_emails":
            spam_emails+=int(line[1])
        elif line[0] == "total_emails":
            total_emails+=int(line[1])
        elif line[0] == 'total_spam_words':
            total_spam_words+=int(line[1])
        elif line[0] == 'total_ham_words':
            total_ham_words+=int(line[1])
        elif line[0] == 'spam_count':
            spam_count+=int(line[1])
        elif line[0] == 'ham_count':
            ham_count+=int(line[1])
        elif line[0] == 'word': #create variable for search word
            word = line[1]
        else: all_emails.append(line)

prior_spam = float(spam_emails)/float(total_emails) #prior spam = s
pam emails / total emails
prior_ham = float(total_emails-spam_emails)/float(total_emails) #pr
ior ham = ham emails / total emails
spam_probability = float(spam_count)/float(total_spam_words) #spam
probability is the number of occurrences of word in spam emails / t
otal words in spam emails
ham_probability = float(ham_count)/float(total_ham_words) #ham prob
ability is the number of occurrences of word in ham emails / total
words in non-spam emails

predictions = []

for email in all_emails:
    count_of_word = 0
```

```
    for each in email:
        if word == each: #create count of word
            count_of_word+=1
    mnb_spam_probability = prior_spam*spam_probability**count_of_wo
rd #formula for calculating probability of spam given a word
    mnb_ham_probability = prior_ham*ham_probability**count_of_word
#formula for calculating probability of ham given a word
    if mnb_spam_probability > mnb_ham_probability: predictions.appe
nd(1) #if probability of spam > ham, prediction of 1 indicates spam
    else: predictions.append(0)

print predictions
```

```
Overwriting reducer.py
```

In [10]:
```
!./pNaiveBayes.sh 5 "assistance"
```

Using "assistance" with my single word MNB model results in the classification of 7 emails in the dataset as spam.

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 [12]:

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


spam_emails = 0
total_emails = 0
total_spam_words = 0
total_ham_words = 0
filename = sys.argv[1]
findwords = sys.argv[2].split(' ')
count_dictionary = {}
count_dictionary['spam'] = {}
count_dictionary['ham'] = {}
emails = open(filename, "r")
for line in emails.readlines():
    line = line.translate(string.maketrans("",""), string.punctuati
on) #strip punctuation
    email = re.split(r'\t+', line)
    if len(email) != 4: #skip over email data formatting errors
        continue
    total_emails+=1
    content = email[0] + email[2] + email[3] #concatenate subject a
nd body sections 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
        spam_emails+=1
        for word in findwords:
            word = word.lower()
            for each in list_content:
                if each.lower() == word:
                    if word not in count_dictionary['spam']: coun
t_dictionary['spam'][word] = 1
                    else: count_dictionary['spam'][word]+=1
                total_spam_words+=1
    else:
        for word in findwords:
            word = word.lower()
            for each in list_content:
                if each.lower() == word:
                    if word not in count_dictionary['ham']: count_d
```

```
ictionary['ham'][word] = 1
                    else: count_dictionary['ham'][word]+=1
                total_ham_words+=1
    print content

print "spam_emails", spam_emails
print "total_emails", total_emails
print "total_spam_words", total_spam_words
print "total_ham_words", total_ham_words
print "word", sys.argv[2]
print count_dictionary
```

```
Overwriting mapper.py
```

In [13]:
```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Jing Xu
## Description: reducer code for HW1.4

import sys
import ast
import math

spam_emails = 0
total_emails = 0
total_spam_words = 0
total_ham_words = 0
words = ''
unique_words = []
all_emails = []
final_count_dictionary = {}
final_count_dictionary['spam'] = {}
final_count_dictionary['ham'] = {}
for filenames in sys.argv[1:]: #open each filename in the countfile
list
    myfile = open('%s'%filenames, "r")
    for line in myfile.readlines():
        if line[0] == "{":
            count_dictionary = line
            count_dictionary = ast.literal_eval(count_dictionary)
#convert dictionary string to dictionary class
            for key in count_dictionary:
                for word in count_dictionary[key]:
                    if word not in final_count_dictionary[key]: fin
al_count_dictionary[key][word] = count_dictionary[key][word]
                    else: final_count_dictionary[key][word] += coun
t_dictionary[key][word]
        else: line = line.split()
        #aggregate counts for all variables of interest
        if line[0] == "spam_emails":
            spam_emails+=int(line[1])
        elif line[0] == "total_emails":
            total_emails+=int(line[1])
        elif line[0] == 'total_spam_words':
            total_spam_words+=int(line[1])
        elif line[0] == 'total_ham_words':
            total_ham_words+=int(line[1])
        elif line[0] == 'word': #create variable for search word
            words = line[1:]
        else: #create list of unique words for later use
            for word in line:
                if word not in unique_words: unique_words.append(wo
rd)
            all_emails.append(line)

prior_spam = float(spam_emails)/float(total_emails) #prior spam = s
```

```
pam emails / total emails
prior_ham = float(total_emails-spam_emails)/float(total_emails) #pr
ior ham = ham emails / total emails

predictions = []

#creation of conditional probability dictionary for all search word
s in all spam and ham emails
conditional_prob = {}
conditional_prob['spam'] = {}
conditional_prob['ham'] = {}
for word in words:
    if word in final_count_dictionary['spam']:
        conditional_prob['spam'][word] = (float(final_count_diction
ary['spam'][word]) + float(1))/(float(total_spam_words) + float(le
n(unique_words)))
    else: conditional_prob['spam'][word] = (float(1))/(float(tota
l_spam_words)+float(len(unique_words)))
    if word in final_count_dictionary['ham']:
        conditional_prob['ham'][word] = (float(final_count_dictiona
ry['ham'][word]) + float(1))/(float(total_ham_words) + float(len(un
ique_words)))
    else: conditional_prob['ham'][word] = (float(1))/(float(total_h
am_words)+float(len(unique_words)))

for email in all_emails:
    mnb_spam_probability = prior_spam #start of MNB formula to calc
ulate ham probability given search words
    mnb_ham_probability = prior_ham #start of MNB formula to calcul
ate ham probability given search words
    for word in words:
        count_of_word = 0
        for each in email:
            if word == each: #create count of word
                count_of_word+=1
            mnb_spam_probability *= float(conditional_prob['spam']
[word]**count_of_word) #completion of formula for calculating proba
bility of spam given a word
            mnb_ham_probability *= float(conditional_prob['ham'][wo
rd]**count_of_word) #completion of formula for calculating probabil
ity of ham given a word
    if mnb_spam_probability > mnb_ham_probability: predictions.appe
nd(1) #if probability of spam > ham, prediction of 1 indicates spam
    else: predictions.append(0)

print predictions
```

```
Overwriting reducer.py
```

In [15]:

```
!./pNaiveBayes.sh 5 "assistance valium enlargementWithATypo"
```

Using "assistance", "valium", and "enlargementWithATypo" with my MNB model results in the classification of 3 emails in the dataset as spam.