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

Big data is a general term for data (sets) either so large or so complex that conventional data processing applications (RMDB etc) are caught short. In machine learning considerations, this term often focus the use of predictive analytics or certain other advanced methods to extract value or learn from generally complex data sets (in volume, volecity, variety, variability, veracity, visulization and value). (For an adnet company I worked for.) As a 300+MM banner-ad-requests+/day ad network company, we were facing a typical big data problem for all volume, velocity and variety considerations. Since the 'problem' graduately developed through out a 12+ year period of time, we ended up with a more traditional solution (co-lo of clusters of 100+ top-of-the-line servers just for the front end requests with 2 hardware load balancer.) Eventually, we used Hadoop/Pig for the preliminary analyitics. (By the time I left in 2012).

HW1.0.1.In 500 words (English or pseudo code or a combination) describe how to estimate the bias, the variance, the irreduciable 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?

First bias is the difference between the expected predicted value and the actual value, or E[y*]−f(x), the variance is the expected squared sum of the difference between the predicted value and the expected predicted value, or E[(y*−E[y*])^2], and the irreducible error (noice) is simply the expected squared sume of the difference between the predicted value and the actual value, or E[(y*−f(x))^2]=E[ε2]=σ^2, which will always be there regardless of the bias and variance pan out. When usig polynomial regression models of degree 1 to 5 for a complex problem dataset, then, the lower the degree (toward 1), the bigger the bias would be, and the smaller the variance [less accurate for tested data but less sensitive to variety of new test data, or underfitting]. On the other hand, the higher the degree (toward 5), the smaller the bias, at the cost of a bigger variance [more accurate for tested data but more sensity to new test data, or overfitting].

I would start from the lowest degeree (1) and work my way up to higher degrees. For each situation, I will check the performance (interms of prediction accuracy with the bench mark because this is going to a supervised learning). I'd work my way up until the performance starts to degregate. Then the most previous degree will be the one I'd pick.

To treat the whole concept more formally and precisely. We have

E[(y-h(x))^2] = E[h(x) = E(h(x))^2] + E(h(x)^2 - 2f(x)E(h(x)) + f(x)^2 + E[(y-f(x))^2] =

E[h(x) - E(h(x))^2] + (this is Variance) (h(x) - f(x))^2 + (this is bias) E[y-f(x))^2] (this is the irreducible error, or noise) = Var[h(x)] + Bias[h(x)]^2 + E[e^2] or = Var[h(x)] + Bias[h(x)]^2 _ sigma^2

For each data point x, we observed value y and some predictions y1 to yk. We just need to compute the expect/mean of predictoin h. We estimate the bias as (h-y) and the variance as sum of (yk - h)^2/(K-1) for all yk, in this case assume noise is 0.

In the example of the sine function using polynomial regression as an example. We'd calculate all three factors against all orders between 1 to 5, we then chose the degree that correspnding to the cross betwen the variance error and bias.

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 [87]:
```
### HW1.1. Read through the provided control script (pNaiveBayes.sh)and all
### purpose and function, respond to the remaining homework questions below
### statmement with  a "done" string will suffice here. (dont forget to inc

print "Done."
```

```
Done.
```

In [88]:

```
%%writefile pNaiveBayes.sh
## pNaiveBayes.sh
## Author: Jake Ryland Williams
## Usage: pNaiveBayes.sh m wordlist
## Input:
##       m = number of processes (maps), e.g., 4
##       wordlist = a space-separated list of words in quotes, e.g., "the a
##
## Instructions: Read this script and its comments closely.
##               Do your best to understand the purpose of each command,
##               and focus on how arguments are supplied to mapper.py/reduc
##               as this will determine how the python scripts take input.
##               When you are comfortable with the unix code below,
##               answer the questions on the LMS for HW1 about the starter

## collect user input
m=$1 ## the number of parallel processes (maps) to run
wordlist=$2 ## if set to "*", then all words are used

## a test set data of 100 messages
data="enronemail_1h.txt"

## the full set of data (33746 messages)
# data="enronemail.txt"

## 'wc' determines the number of lines in the data
## 'perl -pe' regex strips the piped wc output to a number
linesindata=`wc -l $data | perl -pe 's/^.*?(\d+).*?$/$1/'`

## determine the lines per chunk for the desired number of processes
linesinchunk=`echo "$linesindata/$m+1" | bc`

## split the original file into chunks by line
split -l $linesinchunk $data $data.chunk.

## assign python mappers (mapper.py) to the chunks of data
## and emit their output to temporary files
for datachunk in $data.chunk.*; do
    ## feed word list to the python mapper here and redirect STDOUT to a te
    ####
    ####
    ./mapper.py $datachunk "$wordlist" > $datachunk.counts &
    ####
    ####
done
## wait for the mappers to finish their work
wait

## 'ls' makes a list of the temporary count files
## 'perl -pe' regex replaces line breaks with spaces
countfiles=`\ls $data.chunk.*.counts | perl -pe 's/\n/ /'`

## feed the list of countfiles to the python reducer and redirect STDOUT to
####
```

```
####
####
./reducer.py $countfiles > $data.output
####
####

## clean up the data chunks and temporary count files
\rm $data.chunk.*
```

```
Overwriting pNaiveBayes.sh
```

In [89]:
```
!chmod +x pNaiveBayes.sh
```

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 [90]:
```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## author: Alan Wang
## Description mapper.py for HW1.2
## This mapper will take in a file of m (or less) lines and count occurance
## of a single word as its parameter
## then it spit out the word counts in '<word> <count>' format when word is

#First, the mapper.py which will take a file of n lines in and count the nu

import sys
import re
count = 0

WORD_RE = re.compile(r"[\w']+") # regular express for words

if len(sys.argv) != 3:
    print "Usage: mapper.py <file> <word>"
    exit()

fileName = sys.argv[1] #the filename
wordList = re.split(" ", sys.argv[2].lower())  #the wordList for HW1.2 to 1

forAllWords = False

if len(wordList) == 1 and wordList[0] == '*':
    forAllWords = True

wordCounts = {}

with open (fileName, "r") as myFile:
    for line in myFile:
        if forAllWords:
            wordList = WORD_RE.findall(line)
        quar = line.split("\t") # potentially a quardruplet containing id-s
        if len(quar) != 4 or (quar[1] != "0" and quar[1] != '1'): #if there
            continue

        for word in wordList:
            count = WORD_RE.findall(quar[2] + ' ' + quar[3]).count(word)
            if count != 0:
                if word in wordCounts:
                    wordCounts[word] += count
                else:
                    wordCounts[word] = count

if len(wordCounts) != 0:
    print '\n'.join([word + ' ' + str(wordCounts[word]) for word in wordCou
```

```
Overwriting mapper.py
```

In [91]:
```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## author: Alan Wang
## Description reducer.py for HW1.2
## it reads in an aggregated file (from files generated by the mapper.py),

import sys

fileNames = sys.argv[1:] #the filename

wordCounts = {}

for fileName in fileNames:
    with open (fileName, "r") as myFile:
        for line in myFile:
            tuple = line.split(' ')
            word = tuple[0]
            count = tuple[1]
            #print word, count

            if word in wordCounts:
                wordCounts[word] += int(count)
            else:
                wordCounts[word] = int(count)

for word in sorted(wordCounts):
    # in real application where the order does not matter, we might want to
    # to avoid the n*ln n performance penalty
    print word, wordCounts[word]
```

```
Overwriting reducer.py
```

In [92]:
```
!chmod +x mapper.py
!chmod +x reducer.py
!./pNaiveBayes.sh 4 assistance
!tail enronemail_1h.txt.output
```

```
assistance 10
```

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. To do so, make sure that

- mapper.py and
- reducer.py

that performs a single word Naive Bayes classification. For multinomial Naive Bayes, the Pr(X="assistance"|Y=SPAM) is calculated as follows:

the number of times "assistance" occurs in SPAM labeled documents / the number of words in documents labeled SPAM

NOTE if "assistance" occurs 5 times in all of the documents Labeled SPAM, and the length in terms of the number of words in all documents labeld as SPAM (when concatenated) is 1,000. Then Pr(X="assistance"|Y=SPAM) = 5/1000. Note this is a multinomial estimated of the class conditional for a Naive Bayes Classifier. No smoothing is needed in this HW.

In [93]:

```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## author: Alan Wang
## Description mapper.py for HW1.3
## This mapper will take in a file of m (or less) lines and count occurance
## of a single word as its parameter
## then it spit out the word counts in '<flag> <word> <count>' format when

#First, the mapper.py which will take a file of n lines in and count the nu

import sys
import re
count = 0

WORD_RE = re.compile(r"[\w']+") # regular express for words

if len(sys.argv) != 3:
    print "Usage: mapper.py <file> <word>"
    exit()

fileName = sys.argv[1] #the filename
wordList = re.split(" ", sys.argv[2].lower())  #the wordList for HW1.2 to 1

forAllWords = False

if len(wordList) == 1 and wordList[0] == '*':
    forAllWords = True

spamWordCounts = {}
regWordCounts = {}
spamTotal = 0
regTotal = 0

with open (fileName, "r") as myFile:
    for line in myFile:
        if forAllWords:
            wordList = WORD_RE.findall(line)
        quar = line.split("\t") # potentially a quardruplet containing id-s
        if len(quar) != 4 or (quar[1] != "0" and quar[1] != '1'): #if there
            continue

        for word in wordList:
            subTotal = len(WORD_RE.findall(quar[2] + ' ' + quar[3]))
            count = WORD_RE.findall(quar[2] + ' ' + quar[3]).count(word)

            if quar[1] == '1': #spam
                spamTotal += subTotal
                if count !=0:
                    if word in spamWordCounts:
                        spamWordCounts[word] += count
                    else:
                        spamWordCounts[word] = count
            else: #regular
```

```
            else: #regular
                regTotal += subTotal
                if count !=0:
                    if word in regWordCounts:
                        regWordCounts[word] += count
                    else:
                        regWordCounts[word] = count

if len(spamWordCounts) != 0:
    print '\n'.join(['1 ' + word + ' ' + str(spamWordCounts[word]) for word

if len(regWordCounts) != 0:
    print '\n'.join(['0 ' + word + ' ' + str(spamWordCounts[word]) for word

if spamTotal > 0:
    print '1 '+'_total '+str(spamTotal)

if regTotal > 0:
    print '0 '+'_total '+str(regTotal)
```

```
Overwriting mapper.py
```

In [94]:
```
##Naive Baysean:  ln(p(S|D))/p(~S|D)) = ln(P(S)/P(~S)) + sigma(ln(p(wi|S)/p
##first, let's find the prior P(S) and P(~S)
import sys

total = 0
spamTotal = 0

with open('enronemail_1h.txt', 'r') as dataFile:
    for line in dataFile:
        quar = line.split("\t") # potentially a quardruplet containing id-s
        if len(quar) != 4 or (quar[1] != "0" and quar[1] != '1'): #if there
            continue
        total += 1
        if quar[1] == '1':
            spamTotal += 1

print total, spamTotal, float(spamTotal)/total, (1 - float(spamTotal)/total
```

```
100 44 0.44 0.56
```

In [95]:

```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## author: Alan Wang
## Description reducer.py for HW1.3
## it reads in an aggregated file (from files generated by the mapper.py),
## and yield the spam or no spam prediction based upon naive baysean model

import sys
import re
import math

fileNames = sys.argv[1:] #the filename

spamWordCounts = {}
regWordCounts = {}
spamTotal = 0
regTotal = 0

for fileName in fileNames:
    with open (fileName, "r") as myFile:
        for line in myFile:
            triplets = line.split(' ')
            spam = triplets[0]
            word = triplets[1]
            count = triplets[2]
            #print word, count

            if word == '_total':
                if spam == '1':
                    spamTotal += int(count)
                else:
                    regTotal += int(count)
            else:

                if spam == '1':
                    if word in spamWordCounts:
                        spamWordCounts[word] += int(count)
                    else:
                        spamWordCounts[word] = int(count)
                else:
                    if word in regWordCounts:
                        regWordCounts[word] += int(count)
                    else:
                        regWordCounts[word] = int(count)

#At this point we have all the info to calculate the spam-likelyhood for ou
#  ln(p(S|D))/p(~S|D)) = ln(P(S)/P(~S)) + sigma(ln(p(wi|S)/p(wi|~S)))

## From last cell:

pS = 0.44 #prior p(S)
pR = 0.56 #prior p(~S)
pR = math.log(pS/pR) # first term ln(P(S)/P(~S)) the priror ratio
```

```
pR = math.log(pS/pR) # first term in(P(S))P(~S)) the prior ratio

testFileName = 'enronemail_1h.txt'

WORD_RE = re.compile(r"[\w']+") # regular express for words

def probSpam(word):
    c = 0
    if word in spamWordCounts:
        c = spamWordCounts[word]
    return float(c+1)/float(spamTotal + len(spamWordCounts)) #smoothing for

def probReg(word):
    c = 0
    if word in regWordCounts:
        c = regWordCounts[word]
    return float(c+1)/float(regTotal + len(regWordCounts)) #smoothing for b

total = 0
totalCorrect = 0

with open(testFileName, 'r') as testFile:
    for line in testFile:
        quar = line.split("\t") # potentially a quardruplet containing id-s
        if len(quar) != 4 or (quar[1] != "0" and quar[1] != '1'): #if there
            continue

        total += 1
        #otherwise, we perform the 'prediction'
        content = WORD_RE.findall(quar[2] + ' ' + quar[3])

        sigma = pP #initalized to pPR

        for word in content: # for each word
            if word in spamWordCounts and word in regWordCounts:
                ppS = float(spamWordCounts[word])/spamTotal
                ppR = float(regWordCounts[word])/regTotal
                sigma += math.log(ppS/ppR)

        predict = 1

        if(sigma < 0.0):
            predict = 0

        if (sigma >= 0.0 and quar[1] == '1') or (sigma < 0.0 and quar[1] ==
            totalCorrect += 1

        print quar[0], quar[1], predict

print 'Total # of documents:', total, 'Total correct:', totalCorrect, 'Accu
```

```
Overwriting reducer.py
```

In [96]:
```
!chmod +x mapper.py
!chmod +x reducer.py
!./pNaiveBayes.sh 4 assistance
!tail enronemail_1h.txt.output
```

```
0017.1999-12-14.kaminski 0 0
0017.2000-01-17.beck 0 0
0017.2001-04-03.williams 0 0
0017.2003-12-18.GP 1 0
0017.2004-08-01.BG 1 0
0017.2004-08-02.BG 1 0
0018.1999-12-14.kaminski 0 0
0018.2001-07-13.SA_and_HP 1 1
0018.2003-12-18.GP 1 1
Total # of documents: 100 Total correct: 60 Accuracy Rate: 0.6
```

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

To do so, make sure that

- mapper.py counts all occurrences of a list of words, and
- reducer.py

performs the multiple-word multinomial Naive Bayes classification via the chosen list. No smoothing is needed in this HW.

In [97]:

```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## author: Alan Wang
## Description mapper.py for HW1.4
## This mapper will take in a file of m (or less) lines and count occurance
## of a single word as its parameter
## then it spit out the word counts in '<flag> <word> <count>' format when
## Note in my version of the mapper and reducer, there is actually no diffe
## started to envision a wordlist, instead of a single word only in HW1.2 a
## all cases, including the "*" case, which I put at the end.

#First, the mapper.py which will take a file of n lines in and count the nu

import sys
import re
count = 0

WORD_RE = re.compile(r"[\w']+") # regular express for words

if len(sys.argv) != 3:
    print "Usage: mapper.py <file> <word>"
    exit()

fileName = sys.argv[1] #the filename
wordList = re.split(" ", sys.argv[2].lower())  #the wordList for HW1.2 to 1

forAllWords = False

if len(wordList) == 1 and wordList[0] == '*':
    forAllWords = True

spamWordCounts = {}
regWordCounts = {}
spamTotal = 0
regTotal = 0

with open (fileName, "r") as myFile:
    for line in myFile:
        if forAllWords:
            wordList = WORD_RE.findall(line)
        quar = line.split("\t") # potentially a quardruplet containing id-s
        if len(quar) != 4 or (quar[1] != "0" and quar[1] != '1'): #if there
            continue

        for word in wordList:
            subTotal = len(WORD_RE.findall(quar[2] + ' ' + quar[3]))
            count = WORD_RE.findall(quar[2] + ' ' + quar[3]).count(word)

            if quar[1] == '1': #spam
                spamTotal += subTotal
                if count !=0:
                    if word in spamWordCounts:
                        spamWordCounts[word] += count
```

```
                    spamWordCounts[word] += count
                else:
                    spamWordCounts[word] = count
            else: #regular
                regTotal += subTotal
                if count !=0:
                    if word in regWordCounts:
                        regWordCounts[word] += count
                    else:
                        regWordCounts[word] = count

if len(spamWordCounts) != 0:
    print '\n'.join(['1 ' + word + ' ' + str(spamWordCounts[word]) for word

if len(regWordCounts) != 0:
    print '\n'.join(['0 ' + word + ' ' + str(spamWordCounts[word]) for word

if spamTotal > 0:
    print '1 '+'_total '+str(spamTotal)

if regTotal > 0:
    print '0 '+'_total '+str(regTotal)
```

```
Overwriting mapper.py
```

In [98]:

```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## author: Alan Wang
## Description reducer.py for HW1.3
## it reads in an aggregated file (from files generated by the mapper.py),
## and yield the spam or no spam prediction based upon naive baysean model

import sys
import re
import math

fileNames = sys.argv[1:] #the filename

spamWordCounts = {}
regWordCounts = {}
spamTotal = 0
regTotal = 0

for fileName in fileNames:
    with open (fileName, "r") as myFile:
        for line in myFile:
            triplets = line.split(' ')
            spam = triplets[0]
            word = triplets[1]
            count = triplets[2]
            #print word, count

            if word == '_total':
                if spam == '1':
                    spamTotal += int(count)
                else:
                    regTotal += int(count)
            else:

                if spam == '1':
                    if word in spamWordCounts:
                        spamWordCounts[word] += int(count)
                    else:
                        spamWordCounts[word] = int(count)
                else:
                    if word in regWordCounts:
                        regWordCounts[word] += int(count)
                    else:
                        regWordCounts[word] = int(count)

#At this point we have all the info to calculate the spam-likelyhood for ou
#  ln(p(S|D))/p(~S|D)) = ln(P(S)/P(~S)) + sigma(ln(p(wi|S)/p(wi|~S)))

## From last cell:

pS = 0.44 #prior p(S)
pNS = 0.56 #prior p(~S)
pRR = math.log(pS/pNS) # first term ln(P(S)/P(~S))
```

```
pPR = math.log(p3/pN3) # first term ln(P(S)/P(~S))

testFileName = 'enronemail_1h.txt'

WORD_RE = re.compile(r"[\w']+") # regular express for words

def probSpam(word):
    c = 0
    if word in spamWordCounts:
        c = spamWordCounts[word]
    return float(c+1)/float(spamTotal + len(spamWordCounts)) #smoothing for

def probReg(word):
    c = 0
    if word in regWordCounts:
        c = regWordCounts[word]
    return float(c+1)/float(regTotal + len(regWordCounts)) #smoothing for b

total = 0
totalCorrect = 0

with open(testFileName, 'r') as testFile:
    for line in testFile:
        quar = line.split("\t") # potentially a quardruplet containing id-s
        if len(quar) != 4 or (quar[1] != "0" and quar[1] != '1'): #if there
            continue

        total += 1
        #otherwise, we perform the 'prediction'
        content = WORD_RE.findall(quar[2] + ' ' + quar[3])

        sigma = pPR #initalized to pPR

        for word in content: # for each word
            sigma += math.log(probSpam(word)/probReg(word))

        predict = 1

        if(sigma < 0.0):
            predict = 0

        if (sigma >= 0.0 and quar[1] == '1') or (sigma < 0.0 and quar[1] ==
            totalCorrect += 1

        print quar[0], quar[1], predict

print 'Total # of documents:', total, 'Total correct:', totalCorrect, 'Accu
```

```
Overwriting reducer.py
```

In [99]:
```
!chmod +x mapper.py
!chmod +x reducer.py
!./pNaiveBayes.sh 4 "assistance valium enlargementWithATypo"
!tail enronemail_1h.txt.output
```

```
0017.1999-12-14.kaminski 0 0
0017.2000-01-17.beck 0 0
0017.2001-04-03.williams 0 0
0017.2003-12-18.GP 1 0
0017.2004-08-01.BG 1 0
0017.2004-08-02.BG 1 0
0018.1999-12-14.kaminski 0 0
0018.2001-07-13.SA_and_HP 1 0
0018.2003-12-18.GP 1 0
Total # of documents: 100 Total correct: 56 Accuracy Rate: 0.56
```

In [100]:
```
!./pNaiveBayes.sh 4 *
!tail enronemail_1h.txt.output
```

```
0017.1999-12-14.kaminski 0 0
0017.2000-01-17.beck 0 0
0017.2001-04-03.williams 0 0
0017.2003-12-18.GP 1 0
0017.2004-08-01.BG 1 0
0017.2004-08-02.BG 1 0
0018.1999-12-14.kaminski 0 0
0018.2001-07-13.SA_and_HP 1 0
0018.2003-12-18.GP 1 0
Total # of documents: 100 Total correct: 56 Accuracy Rate: 0.56
```

In [ ]:
```

```