MIDS-W261-2015-HWK-Week01-Spooner

HW 1.0.0 Define big data. Provide an example of a big data problem in your domain of expertise.

Big Data: Big data is the study and discipline of knowledge at the extreme large end of variety, velocity, or volume. Often two or all of these qualities will exist to a large degree. The exact size will vary over time as the term is relative to what is considered "normal sized" data. However, currently one might roughly think about big data variety in the dozens to hundreds of sources, big data velocity in terms of terrabytes per second, and big data volume in terms of petabytes.

In my work domain of HMOs big data lies at the intersection of social media data being merged into health data, or live monitoring data such as blood pressure and heart rate into disease managment data. In these cases you take the already terrabytes of health data stored from insurance claims and are merging them with an enormous volume and velocity of streaming live data for the purpose of desease managment or near real time coordinated care services.

HW 1.0.1 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?

Bias and variance are usually at odds. Bias is the difference between your model and the true model. Variance is the difference between your model and the data. If you reduce variance you'll over fit and introduce bias. If you try to fit your model to a idealized theoretical model which closely matches the true underlying function then you'll probably have to live with some variance as the data will rarely match 100%.

Bias and variance can be estimated by looking at the difference between the regression models of 1 degree and the models of 5 degrees. A 5 degree model should have lower variance than bias. Depending on what assumptions you make about the true underlying function you might be able to conclude that the one degree model has less bias despite higher variance.

Much of this is guesswork but looking at visual representations of the model and running sub-regressions may help deduce the underlying function. By choosing a model of the correct degrees that matches the general trend in the underlying data or intuitively matches the behavior of the underlying you can increase your chances of reducing both bias and variance.

The most powerful tool however is using test and training data. Models trained on the training data are tested against the test data. Over fit models will have high variance when measured against the test data. Biased models will perform consistently off from the training data, although depending on the complexity of your data and model this consistent bias may be hard to isolate. Simplifying the data or isolating individual variables or interaction terms may permit you to find hidden trends not captured by your model that would create bias.

Irreducible Error is the noise inherent in the data or measurement. Even the best most perfect model would never be able to get rid of this error, thus the "irreducible" part. Irreducible error can be estimated by calibrating your measurement devices against known standards. It can also be estimated by generating a high confidence model with simplified data and finding the variance remaining in this simplified high-confidence set. Your best model measured against the training data with best performance is your best way to estimate irreducible error in the absence of calibration.

Choosing the best Model: Use test and training data. Depending on the size of your data set a half and half split may be appropriate. I'd want to save no less than 20% for test data. I'd begin by plotting the data and trying to get a general idea about its trends and shape. Then i would apply a function to train 5 polynomial regression models of degree 1 - 5. Then I'd test these models against the test data and examine their summary statistics in addition to plotting them to examine best fit. By comparing the test performance to the training performance I could estimate bias. If I had a very well fitting model which matches the trends in data when plotted I may be able to estimate the irreducible error by looking at the remaining test variance when using this model.

HW 1.1 Read through the provided control script (pNaiveBayes.sh)

Done

HW 1.2 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 [53]:

```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Spooner
```

```
## Description: mapper code for HW1.2

import sys
import re
count = 0

## collect user input
filename = sys.argv[1]
findwords = re.split(" ",sys.argv[2].lower())
f = open(filename).read()
words = re.split('\s', f)
count = 0
for each in words:
    if each in findwords:
        count = count + 1

print count
```

```
Overwriting mapper.py
```

In [133]:

```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Spooner
## Description: mapper code for HW1.2

import sys
import re
count = 0

## collect mapper output and sum
s = 0

for each in range(1,len(sys.argv)):
    f = open(sys.argv[each])
    s = s + int(f.read())
print s
```

```
Overwriting reducer.py
```

In [96]:

```
#change permissions
!chmod +x mapper.py; chmod +x reducer.py
```

```
'chmod' is not recognized as an internal or external command,
operable program or batch file.
```

In [136]:

```
#run pNaiveBayes.sh
!pNaiveBayes.sh 5 assistance

f = open("enronemail_1h.txt.output")
print "Mapper output: " + f.read()
```

```
Welcome to Git (version 1.8.4-preview20130916)


Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.
Mapper output: 6
```

**HW 1.3 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 [211]:

```
%%writefile mapper.py
#!/usr/bin/python
## mapper.py
## Author: Spooner
## Description: mapper code for HW1.3

import sys
```

```
import sys
import re
count = 0

## collect user input
filename = sys.argv[1]
findwords = re.split(" ",sys.argv[2].lower())
f = open(filename).readlines()

d = dict((key, [0,0,0,0]) for key in findwords)

for key in d:
    for line in f:
        sets = re.split('\t',line.lower())
        #skip malformed lines
        if len(sets) != 4:
            continue
        if sets[1] == '1':
            #store total words in email
            d[key][0] = d[key][0] + len(sets[2].split()) + len(sets[3].split())
            for word in re.split(' ', sets[2]+sets[3]):
                if word == key:
                    d[key][1] = d[key][1] + 1
        else:
            d[key][2] = d[key][2] + len(sets[2].split()) + len(sets[3].split())
            for word in re.split(' ', sets[2]+sets[3]):
                if word in findwords:
                    d[key][3] = d[key][3] + 1


print d
```

```
Overwriting mapper.py
```

In [ ]:

```
#testing parsing
import re
f = open("enronemail_1h.txt").readlines()
for line in f:
    sets = re.split('\t',line)
    print sets
```

In [216]:

```
%%writefile reducer.py
#!/usr/bin/python
## reducer.py
## Author: Spooner
## Description: mapper code for HW1.3

import sys
import re
import ast
count = 0

## collect mapper output and sum
out = {}

for each in range(1,len(sys.argv)):
    f = open(sys.argv[each])
    s = f.read()

    d = ast.literal_eval(s)
    for key in d:
        if key not in out:
            out[key] = [0,0,0,0]
        out[key][0] = out[key][0] + d[key][0]
        out[key][1] = out[key][1] + d[key][1]
        out[key][2] = out[key][2] + d[key][2]
        out[key][3] = out[key][3] + d[key][3]

msg = ''
for key in out:
    msg = msg + '\n' + key +': given spam: ' + str(out[key][1]*1.0/out[key][0]) + '\n' + key +': given
not spam: ' + str(out[key][3]*1.0/out[key][2])
    msg = msg + '\n' + str(out[key])
print msg
```

```
print msg
```

```
Overwriting reducer.py
```

In [181]:

```
# testing reducer parsing
import ast
f = open("enronemail_1h.txt.chunk.aa.counts").read()
d = ast.literal_eval(f)
for key in d:
    print d[key]
```

```
[860, 0, 2894, 0]
```

In [217]:

```
#run pNaiveBayes.sh
!pNaiveBayes.sh 5 "assistance"

f = open("enronemail_1h.txt.output")
print "Mapper output: " + f.read()
```

```
Welcome to Git (version 1.8.4-preview20130916)


Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.
Mapper output:
assistance: given spam: 0.000263490725126
assistance: given not spam: 7.17566016073e-05
[18976, 5, 13936, 1]
```

HW 1.4 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 [218]:

```
#I programmed the HW 1.3 code to work with multiple arguments
!pNaiveBayes.sh 5 "assistance valium enlargementWithATypo"

f = open("enronemail_1h.txt.output")
print "Mapper output: " + f.read()
```

```
Welcome to Git (version 1.8.4-preview20130916)


Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.
Mapper output:
assistance: given spam: 0.000263490725126
assistance: given not spam: 7.17566016073e-05
[18976, 5, 13936, 1]
enlargementwithatypo: given spam: 0.0
enlargementwithatypo: given not spam: 7.17566016073e-05
[18976, 0, 13936, 1]
valium: given spam: 0.0
valium: given not spam: 7.17566016073e-05
[18976, 0, 13936, 1]
```

HW 1.5 Provide a mapper/reducer pair that, when executed by pNaiveBayes.sh will classify the email messages by all words present.

Cancelled as per email from Instructor. Smoothing covered in lesson 2.

HW 1.6 Benchmark your code with the Python SciKit-Learn implementation of multinomial Naive Bayes

Cancelled as per email from Instructor. Smoothing covered in lesson 2.

In [ ]:

```
```