DATASCI W261: Machine Learning at Scale

David Rose
david.rose@berkeley.edu
W261-1
Week 01
2015.08.31

HW1.0.0

Everyone seems to have their own definition of big data, the 3 Vs, etc. Here's another: big data is information of sufficient size and complexity to require a new and different set of tools and techniques to effectively make use of it as compared to traditional data processing. A orollary of this definition is that in the near future what is considered big data will no longer be, since the tools and techniques will have become the new normal.

The human genome data is an example of big data. Genomic information will play an increasingly large role in healthcare and population health in the future. In terms of size the 1000 Genomes Project contains more than 200 TB of data, for example.

HW1.0.1

Let n be the number of polynomial regression models to be considered.

Let m_n be the n^{th} model with polynomial degree n.

Let j be the number of records in dataset T.

Let k be the number of desired subsets of T to be used for training and testing.

Divide T into k subsets containing (j/k) records in each, such that T_k is the j^{th} subset of T.

For each model m_p do

  For each data subset T_q do

    Divide T_q into two non-overlapping subsets of equal size, $T_q{}^{train}$ and $T_q{}^{test}$, to be used for training and testing model m_p

    Train model m_p on data $T_q{}^{train}$

For each tuple [x,y] in T_q^{test}

Calculate $m_p(x)$; store $m_p(x)$ and y for subsequent calculations

For each model calculate estimated average bias, variance and prediction error:

$bias^2$ = average of the average squared difference between $m_p(x)$ and y for each T_q^{test}

variance = average difference between $m_p(x)$ and the average value of $m_p(x)$ for all datasets T_q^{test}

prediction error = $bias^2$ + variance plus a constant representing noise for each T_q^{test}. The constant is ignored since it is, after all, constant.

The best model is selected by determining the model with the minimal prediction error.

HW1.1

In [71]:

```
#HW1.1. Read through the provided control script (pNaiveBayes.sh)
!printf 'done'
```

```
done
```

Mapper script. The same mapper is used for each subsequent exercise. Of note:

- both email subject and email body are considered together
- in addition to emitting word counts the mapper also emits email classification counts
    - this is not consistent with functional programming, but here it's okay

In [58]:

```
%%writefile mapper.py
#!/usr/bin/python
''' mapper reads name of file containing chunk of email records an
    words of interest

    mapper emits counts of words, and counts of email classificati
'''
from __future__ import print_function
import re
import string
import sys
filename = sys.argv[1]
findwords = sys.argv[2].split()
# regular expression to remove all punctuation
punctuation = re.compile('[%s]' % re.escape(string.punctuation))
with open (filename, "r") as myfile:
    for line in myfile:
        # split line into three tokens: id, classification, email
        # both the email subject and the email body are included i
        tokens = line.split('\t', 2)
        isspam = tokens[1]
        # emit count of email classification, using magic word '__
        if isspam == '0': # ham
            print('__CLASS__', 1, 0)
        else: # spam
            print('__CLASS__', 0, 1)
        # convert text to lower case
        text = tokens[len(tokens) - 1].lower()
        # remove punctuation from text
        text = punctuation.sub('', text)
        # split into individual words
        words = re.findall(r"[\w']+", text)
        for word in words:
            # only report on word if it is in the word list parame
            # or report on all words if parameter equals '*'
            if word in findwords or sys.argv[2] == '*':
                # emit the word and the classification count
                if isspam == '0': # ham
                    print(word, 1, 0)
                else: # spam
                    print(word, 0, 1)

```

Overwriting mapper.py

Reducer script for HW1.2. Aggregates results from mapper, emits summed counts of all words processed by mapper.

In [59]:

```
%%writefile reducer.py
#!/usr/bin/python
''' reducer is provided list of temporary files containing mapper

    reducer reads each file and aggregates counts of words, them e
'''
from __future__ import print_function
import sys
filelist = sys.argv
words = {}
while len(filelist) > 1: # do not use sys.argv[0]
    with open(filelist.pop(), 'r') as cfile:
        for line in cfile:
            tokens = line.split()
            word = tokens[0]
            if word not in words.keys():
                words[word] = 0
            # mapper produces counts based on email classification
            # this reducer is only interested in total counts
            words[word] += int(tokens[1]) + int(tokens[2])
# emit results
for word in sorted(words.keys()):
    # ignore counts for email classification
    if word != '__CLASS__':
        print('\t'.join([word, str(words[word])]), file=sys.stderr
        print('\t'.join([word, str(words[word])]))

```

```
Overwriting reducer.py
```

HW1.2

In [60]:

```
# HW1.2. Provide a mapper/reducer pair that, when executed by pNai
# will determine the number of occurrences of a single, user-speci
!chmod +x *.py
!./pNaiveBayes.sh 4 "assistance"
```

```
assistance      10
```

Reducer script for HW1.3-5. Aggregates results from mapper to generate vocabulary and email classification counts.

Test the results against the same data set, classifying email records and comparing the results to known classifications.

NOTE: executing the next cell will overwrite the reducer script created in the earlier cell

In [61]:

```
%%writefile reducer.py
#!/usr/bin/python
''' reducer is provided a list of temporary files containing mappe

```

```
    reducer reads each file and aggregates counts of words and ema

    reducer then applies a Naive Bayes classifier against the same
    to buld the training parameters, classifying emal records and
    results to known classsifications.
'''
from __future__ import print_function
import math
import re
import string
import sys
# store statistics on the original list of words of interest
keywords = {}
# counts of each email classification
hamcount = 0
spamcount = 0
# counts of words in each email classification
spamwordcount = 0
hamwordcount = 0

filelist = sys.argv
while len(filelist) > 1:
    with open(filelist.pop(), 'r') as cfile:
        for line in cfile:
            tokens = line.split()
            word = tokens[0]
            # special case for count of email classification
            if word == '__CLASS__':
                hamcount += int(tokens[1])
                spamcount += int(tokens[2])
            # regular case of count of word
            else:
                if word not in keywords.keys():
                    keywords[word] = [0, 0]
                keywords[word][0] += int(tokens[1])
                keywords[word][1] += int(tokens[2])
                hamwordcount += int(tokens[1])
                spamwordcount += int(tokens[2])
# total number of unique words
vocabcount = len(keywords)
# total number of email records
doccount = spamcount + hamcount

# counters for determining error rate
correct = 0
incorrect = 0

# regular expression for removing punctuation
punctuation = re.compile('[%s]' % re.escape(string.punctuation))
with open('enronemail_1h.txt', 'r') as cfile:
    for line in cfile:
        # words to be used in Naive Bayes classification
        nbwords = {}
        tokens = line.split('\t', 2)
```

```
        eid = tokens[0]
        isspam = tokens[1]
        # build bag of words for email record
        text = tokens[len(tokens) - 1].lower()
        text = punctuation.sub('', text)
        docwords = re.findall(r"\w+", text)
        for word in docwords:
            if word in keywords.keys():
                if word not in nbwords:
                    nbwords[word] = 1
                else:
                    nbwords[word] += 1

        # calculate the probability of the email record being spa
        # natural log conversion is used to avoid floating point 

        # start with the prior probability of a spam record
        logpspam = math.log(spamcount / float(doccount))
        for word in nbwords:
            # add the probability of the word being present in th
            # multiplied by the number of times the word appears 
            logpspam += (nbwords[word] *
                (math.log(keywords[word][1] + 1 / float(spamwordc

        # start with the prior probability of a ham record
        logpham = math.log(hamcount / float(doccount))
        for word in nbwords:
            # add the probability of the word being present in th
            # multiplied by the number of times the word appears 
            logpham += (nbwords[word] * (math.log(keywords[word][

        # determine the classification, based on comparison of lo
        nbclass = '0'
        if logpspam > logpham:
            nbclass = '1'

        # add some statistics
        if isspam == nbclass:
            correct += 1
        else:
            incorrect += 1

        # emit the results
        #print('\t'.join([eid, isspam, nbclass, str(isspam == nbc
        print('\t'.join([eid, isspam, nbclass]))
# print some statistics
print('correct: {}, incorrect: {}, training error: {}'.format(cor
    str(float(incorrect) / (correct + incorrect))), file=sys.stde


```

```
Overwriting reducer.py
```

In [62]:

```
!chmod +x *.py
```

HW1.3

In [63]:

```
# HW1.3. Provide a mapper/reducer pair that, when executed by pNai
# will classify the email messages by a single, user-specified wor
# using the Naive Bayes Formulation.
!./pNaiveBayes.sh 4 "assistance"
```

```
correct: 60, incorrect: 40, training error: 0.4
```

HW1.4

In [64]:

```
# HW1.4. Provide a mapper/reducer pair that, when executed by pNai
# will classify the email messages by a list of one or more user-s
!./pNaiveBayes.sh 4 "assistance valium enlargementWithATypo"
```

```
correct: 63, incorrect: 37, training error: 0.37
```

HW1.5

In [65]:

```
# HW1.5. Provide a mapper/reducer pair that, when executed by pNai
# will classify the email messages by all words present.
!./pNaiveBayes.sh 4 "*"
```

```
correct: 100, incorrect: 0, training error: 0.0
```

The benchmark script compares performance (in terms of error rates, not execution time) of the SciKit-Learn implementations of the Multinomial Naive Bayes algorithm and the Bernoulli Naive Bayes algorithm.

In [66]:

```
%%writefile benchmark.py
#!/Users/david/anaconda/bin/python
from __future__ import print_function
import re
import string
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
import sys
records = []
labels = []
# regular expression for removing punctuation
punctuation = re.compile('[%s]' % re.escape(string.punctuation))

# read the input data and create separate lists for content and cl
with open('enronemail_1h.txt', 'r') as cfile:
    for line in cfile:
        tokens = line.split('\t', 2)
        eid = tokens[0]
        label = tokens[1]
        # prepare text
        text = tokens[len(tokens) - 1].lower()
        text = punctuation.sub('', text)
        records.append(text) # content
        labels.append(label) # classification
# prepare the features, using the SciKit-Learn CountVectorizer
data = CountVectorizer().fit_transform(records)

# train and test using the Multinmial Naive Bayes implemenation
clf = MultinomialNB()
clf.fit(data, labels)
results = clf.predict(data)
# measure and report training error
incorrect = 0
for a,b in zip(labels, results):
    incorrect += not a == b
print('Multinomial NB Training Error: ', str(float(incorrect) / le

# train and test using the Multinmial Naive Bayes implemenation
clf = BernoulliNB()
clf.fit(data, labels)
results = clf.predict(data)
# measure and report training error
incorrect = 0
for a,b in zip(labels, results):
    incorrect += not a == b
print('Bernoulli NB Training Error:   ', str(float(incorrect) / le
```

```
Overwriting benchmark.py
```

HW1.6

In [73]:

```
# HW1.6 Benchmark your code with the Python SciKit-Learn implement
!chmod +x *py
!./benchmark.py
!printf 'HW1.5 Training Error: ' && ./pNaiveBayes.sh 4 "*"
```

```
Multinomial NB Training Error:  0.0
Bernoulli NB Training Error:    0.21
HW1.5 Training Error: correct: 100, incorrect: 0, training error:
0.0
```

Results:

Model Type	Training Error
Multinomial NB	0.0
Bernoulli NB	0.21
HW1.5	0.0

Discussion: There are no differences in the results between the SciKit-Learn Multinomial Naive Bayes implementation and the HW1.5 implementation. Since both are training and testing over the same data set it is not surprising that both achieve a training error rate of 0.0.

As seen in the table above, the SciKit-Learn Bernoulli Naive Bayes implementation did not perform as well as the Multinomial Naive Bayes implemenation. This can be ascribed to the fact that the Bernoulli approach uses a dichotomous value for the presence or absence of a term in an email record, whereas the Multinomial approach takes into consideration the number of times a term occurs, yielding a more accurate representation.