MIDS-W261-2016-HWK-Week01-Krishnaswami

January 21, 2016

Natarajan Krishnaswami
W261 Machine Learning At Scale
Spring 2016 / Section 2
Homework 1, Week 2 17 Jan 2016

1 HW1.0.0.

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

Answer: Big data problems are where data is too copious (volume), rapidly arriving or changing (velocity), or nonuniform (variety) to be processed effectively on a single commodity machine (single digit CPUs, single-to-low-double digit GB of RAM, and single digit TB of disks) or small number of them.

In financial information processing, data can come in realtime from funds, index providers, exchanges, customers, etc., in a multitude of formats. These can be price or price-derived indicators (columns), corporate fundamental data scraped from disclosures, or bond/derivative characteristics scraped from contractual terms and conditions. These all need to be normalized, identified, and stored efficiently (columnar stores are an excellent fit for historical time-series, eg).

In addition to the usual three V's, data quality (veracity) is critical to having a trustworthy product. Some of the ways this affects system design incude, for the slower-updating streams like index open/close values, staging/versioning to permit sanity checks (human and automated). For equities, prior revisions of estimates (e.g., analyst target for 52 week earnings) are retained to form a kind of spiky or branching time series. (These two differ in that the former are corrections, and the latter updates; usually people are not as interested in incorrect data, except to explain errors in downstream processing.) For any data type, if erroneous values were calculated or acquired, we must be able to revise them and then regenerate consistent derived data.

2 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?

Answer: I would start by plotting the data and the fit curves for each degree (with dimensionality reduced appropriately, if needed) to see if I can get an rough sense of the shape/likely biases.

One first idea could be to estimate the variance using resampled versions of the training dataset; we expect this to increase with the polynomial degree. We expect to see the error due to squared bias decrease with polynomial degree, but since we don't know the true values, we can do pretty well just by keeping the irreducible error and bias combined.

An easier way to approach this is to note that the tradeoff corresponds precisely to generalizability, so reserving some of the data for testing trained models (or better, doing so repeatedly as in N-fold cross validation) will let us calculate a plausible stand-in for test MSE, provided we have enough data:

Error in the training data suggests the extent of error due to squared bias and irreducible error, and the magnitude of increase in error between training and test data sets suggests the extent of error due to variance.

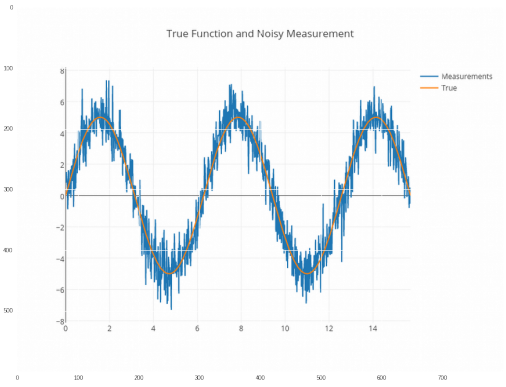
Below, I synthesize some data and illustrate the approach.

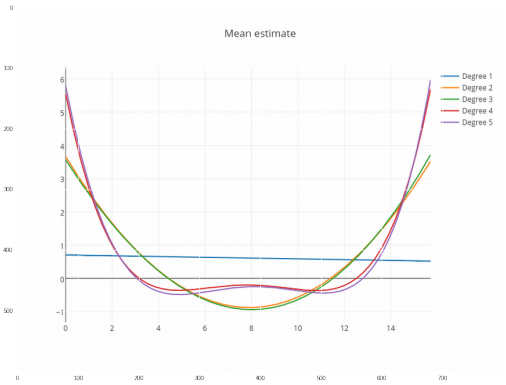
In [479]:
```
%matplotlib inline
import numpy as np
#import plotly.offline as py
#py.init_notebook_mode()
import plotly.plotly as py
import plotly.graph_objs as go
import plotly.tools as tls
import seaborn

from sklearn.utils import resample
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
```

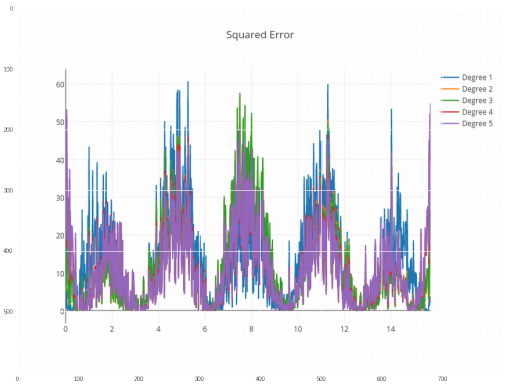
In [480]:
```
def gen_data(min_x, max_x, count, seed=None):
    """generate some data"""
    X=np.linspace(min_x, max_x,count)
    y_true=5*np.sin(X)
    if seed:
        np.random.seed=seed
    noise=np.random.normal(0,1,count)
    y_meas=y_true+noise
    return X,y_true,y_meas

def vary_predict(X, y, deg, n_samples):
    """fit a model of specified degree after resampling
    returns prediction on full data
    """
    rs=resample(np.vstack((X,y)).T, n_samples=n_samples)
    X_rs,y_rs=rs[:,0].reshape(-1,1), rs[:,1].reshape(-1,1)
    pl=Pipeline([('pf', PolynomialFeatures(degree=deg)),
                 ('lr', LinearRegression())])
    pl.fit(X_rs,y_rs)
    return pl.predict(X.reshape(-1,1))

def mean_fit_degree(X, y, deg, n_samples, num_reps):
    """fit many models of specified degree after resampling
    returns avg prediction on full data
    """
    preds=[]
    for rep in range(0,num_reps):
        preds.append(
            vary_predict(X, y, deg, n_samples).flatten())
    preds=np.asanyarray(preds)
    return np.mean(preds, axis=0), np.var(preds, axis=0)
```

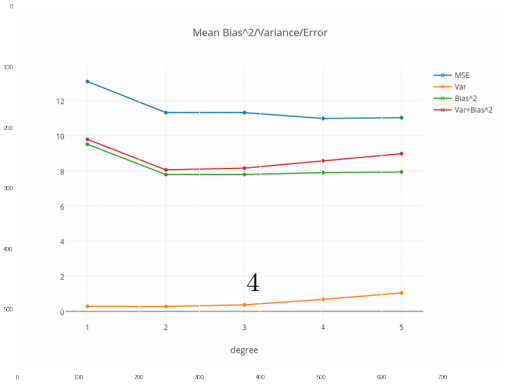
In [501]:
```
def make_plots(num_degs=6,
               n_data=1000,
               n_samples=100,
               n_resamples=200):
    #output variables
    sc=[]
    sqe=[]
```

```
    mse=[]
    mvar=[]
    bias2=[]

    # get data
    X,y_true,y_meas=gen_data(0,5*np.pi, n_data, seed=0)

    # fit some models
    degs=range(1,num_degs)
    for deg in degs:
        y_mean, y_var=mean_fit_degree(X, y_meas, deg, n_samples, n_resamples)
        sc.append(go.Scatter(x=X, y=y_mean, name="Degree "+str(deg)))
        sqe.append(go.Scatter(x=X, y=(y_meas-y_mean)**2, name="Degree "+str(deg)))
        mse.append(((y_meas-y_mean)**2).mean())
        mvar.append(y_var.mean())
        bias2.append(np.abs(y_true-y_mean).mean()**2)

    # plot the data
    py.iplot(
        go.Figure(
            data=go.Data([go.Scatter(x=X, y=y_meas, name="Measurements"),
                          go.Scatter(x=X, y=y_true, name="True"),]),
            layout=go.Layout(title='True Function and Noisy Measurement')),
        show_link=False)

    # plot the output
    mvar=np.asanyarray(mvar)
    py.iplot(go.Figure(data=sc, layout=go.Layout(title="Mean estimate")),
        show_link=False)
    py.iplot(go.Figure(data=sqe, layout=go.Layout(title="Squared Error")),
        show_link=False)
    py.iplot(
        go.Figure(
            data=go.Data([go.Scatter(x=degs, y=mse, name="MSE"),
                          go.Scatter(x=degs, y=mvar, name="Var"),
                          go.Scatter(x=degs, y=bias2, name="Bias^2"),
                          go.Scatter(x=degs, y=mvar+bias2, name="Var+Bias^2"),]),
            layout=go.Layout(title="Mean Bias^2/Variance/Error",
                             xaxis={'title':'degree'})),
         show_link=False)
make_plots()
```


True Function and Noisy Measurement
Measurements
True


Mean estimate
Degree 1
Degree 2
Degree 3
Degree 4
Degree 5


Squared Error
Degree 1
Degree 2
Degree 3
Degree 4
Degree 5


Mean Bias^2/Variance/Error
MSE
Var
Bias^2
Var+Bias^2
degree

3 Instructions/Goals

The data you will use is a curated subset of the Enron email corpus (whose details you may find in the file `enronemail_README.txt` in the directory surrounding these instructions).

In this directory you will also find starter code (`pNaiveBayes.sh`), (similar to the `pGrepCount.sh` code that was presented in this weeks lectures), which will be used as control script to a python mapper and reducer that you will supply at several stages. Doing some exploratory data analysis you will see (with this very small dataset) the following:

```
In [1]: !wget -O pNaiveBayes.sh 'https://www.dropbox.com/sh/jylzkmauxkostck/AAAHAYB6SvwiGpMtJu_04mYaa/pN
        !wget -O enronemail_1h.txt 'https://www.dropbox.com/sh/jylzkmauxkostck/AAC_6JZH7yqMcxfEGPc4-_xJa
        # some cleaning
        import os
        import re
        with open('enronemail_1h.txt.new','w') as out:
            with open('enronemail_1h.txt','rU') as f:
                for line in f:
                    # fix line termination
                    line=line.strip()
                    # fix line 4
                    line=re.sub('^0001.2000-06-06.lokay\t0\t" key dates and impact of upcoming sap imple
                                '0001.2000-06-06.lokay\t0\tkey dates and impact of upcoming sap impleme
                                line)
                    # fix line 50
                    line=re.sub('^0009.2001-06-26.SA_and_HP\t1\t"double',
                                '0009.2001-06-26.SA_and_HP\t1\tNA\t"double',
                                line)
                    # print emits a newline
                    print >>out, line
        os.rename('enronemail_1h.txt.new', 'enronemail_1h.txt')
```

```
--2016-01-18 19:36:02--  https://www.dropbox.com/sh/jylzkmauxkostck/AAAHAYB6SvwiGpMtJu_04mYaa/pNaiveBaye
Resolving www.dropbox.com (www.dropbox.com)... 108.160.172.206, 108.160.172.238
Connecting to www.dropbox.com (www.dropbox.com)|108.160.172.206|:443... connected.
HTTP request sent, awaiting response... 302 FOUND
Location: https://dl.dropboxusercontent.com/content_link/Gyz69W1UoLSu4ZDTioeqCJ0R6vINxJ8GJG6BPoa8KfMRi5r
--2016-01-18 19:36:02--  https://dl.dropboxusercontent.com/content_link/Gyz69W1UoLSu4ZDTioeqCJ0R6vINxJ8G
Resolving dl.dropboxusercontent.com (dl.dropboxusercontent.com)... 45.58.74.5
Connecting to dl.dropboxusercontent.com (dl.dropboxusercontent.com)|45.58.74.5|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2070 (2.0K) [text/x-sh]
Saving to: 'pNaiveBayes.sh'

pNaiveBayes.sh      100%[====================>]   2.02K  --.-KB/s   in 0s

2016-01-18 19:36:03 (221 MB/s) - 'pNaiveBayes.sh' saved [2070/2070]

--2016-01-18 19:36:03--  https://www.dropbox.com/sh/jylzkmauxkostck/AAC_6JZH7yqMcxfEGPc4-_xJa/enronemail
Resolving www.dropbox.com (www.dropbox.com)... 108.160.172.238, 108.160.172.206
Connecting to www.dropbox.com (www.dropbox.com)|108.160.172.238|:443... connected.
HTTP request sent, awaiting response... 302 FOUND
```

```
Location: https://dl.dropboxusercontent.com/content_link/3aOH2eWISZrsgsXLRzcjhcLrtw6GYVfm4HyquJmXiujfYQk
--2016-01-18 19:36:04--  https://dl.dropboxusercontent.com/content_link/3aOH2eWISZrsgsXLRzcjhcLrtw6GYVfm
Resolving dl.dropboxusercontent.com (dl.dropboxusercontent.com)... 108.160.173.5
Connecting to dl.dropboxusercontent.com (dl.dropboxusercontent.com)|108.160.173.5|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 204579 (200K) [text/plain]
Saving to: 'enronemail_1h.txt'

enronemail_1h.txt   100%[====================>] 199.78K  --.-KB/s   in 0.05s

2016-01-18 19:36:04 (3.62 MB/s) - 'enronemail_1h.txt' saved [204579/204579]
```

```
In [2]: !set -x; wc -l enronemail_1h.txt  #100 email records
        !set -x; cut -f2 -d'        ' enronemail_1h.txt|wc  #extract second field which is SPAM flag
        !set -x; cut -f2 -d'        ' enronemail_1h.txt|head
        !set -x; head -n 100 enronemail_1h.txt|tail -1  #an example SPAM email record
```

```
+ wc -l enronemail_1h.txt
100 enronemail_1h.txt
+ cut -f2 -d         enronemail_1h.txt
+ wc
    100     100     200
+ cut -f2 -d         enronemail_1h.txt
+ head
0
0
0
0
0
0
0
0
1
1
+ head -n 100 enronemail_1h.txt
+ tail -1
0018.2003-12-18.GP        1         await your response        " dear partner,  we are a team of governm
```

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. (dont forget to include the Question Number and the quesition in the cell as a multiline comment!)

- Done.
  N.b., instead of including the problem numbers as comments in the code cells, I've formatted this notebook to use the question numbers as headings.

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.

To do so, make sure that

- mapper.py counts all occurrences of a single word, and
- reducer.py collates the counts of the single word.

CROSSCHECK:
bash $ grep assistance enronemail_1h.txt|cut -d"‘printf '\t'‘" -f4| grep assistance|wc -l 8

“assistance” occurs on 8 lines but how many times does the token occur? 10 times! This is the number we are looking for!

In [3]:
```
%%writefile mapper.py
#!/usr/bin/env python
import re
import sys

if len(sys.argv) != 3:
    print >>sys.stderr, "Usage: {0} [filename] [word]".format(sys.argv[0])
    sys.exit(1)
count=0
linenum=0
str_re=re.compile(r'\w+')
with open(sys.argv[1]) as f:
    tgt=sys.argv[2].lower()
    for linenum, line in enumerate(f,1):
        fields=line.lower().split('\t')
        if len(fields) != 4:
            print >>sys.stderr, "Line {0}: expected 4 fields, found {1}".format(linenum, len(fi
            continue
        for word in str_re.findall(''.join(fields[-2:])):
            if word == tgt:
                count += 1
print tgt, count
sys.exit(0)
```

```
Overwriting mapper.py
```

In [4]:
```
%%writefile reducer.py
#!/usr/bin/env python
from collections import Counter
import sys

if len(sys.argv) == 1:
    print >>sys.stderr, "Usage: {0} [filenames]*".format(sys.argv[0])
    sys.exit(1)
# Counter to avoid ugly key existence check in the loop
# This is overkill here, but needed if the reducers could count multiple words
counts = Counter()
for part_file in sys.argv[1:]:
    with open(part_file) as f:
        for linenum, line in enumerate(f,1):
            words=line.strip().split()
            counts[words[0]] += int(words[1])
for word in sorted(counts):
    print >>sys.stderr, '{0}\t{1}'.format(word, counts[word])
```

```
            print '{0}\t{1}'.format(word, counts[word])
        sys.exit(0)

Overwriting reducer.py
```

```
In [5]: !chmod +x mapper.py reducer.py pNaiveBayes.sh
        !./pNaiveBayes.sh 1 assistance
        !grep -i assistance enronemail_1h.txt | cut -d"`printf '\t'`" -f3,4 | tr ' ' '\n' | grep -i ass
        !./pNaiveBayes.sh 1 copyright
        !grep -i copyright enronemail_1h.txt | cut -d"`printf '\t'`" -f3,4 | tr ' ' '\n' | grep -i copy
```

```
assistance        10
10
copyright        7
7
```

Addendum: The data errors described below were fixed in updated data for the assigment

Note: There were some malformed lines (too few fields) in the file, which I corrected manually. I added diagnostic code to print the line numbers after the initial run threw an exception at `fields[-2:]` for a line with a single column.

Line 4: The subject and body were concatenated (space instead of tab). I changed a space to a tab in a reasonable spot to look like a subject and body:
> `key dates and impact of upcoming sap implementation[ ]over the next few weeks`

Line 50: The subject and body were concatenated (space instead of tab). I didn't see a great spot to break the two, so I added a dummy subject: >`[???  ]double your life`

Line 60: A word in the body was replaced by a newline. I joined this and the subsequent line using a placeholder for the missing word:
> `every person was running for his life. my [???] and i managed to escape to south africa`

Since the current assignment weighs subject and body equally, and only word characters are considered, the first two do not alter the probabilities. The third is a trickier choice since the obvious approaches (concatenating the line with the prior line vs omitting the row) yield different probabilities for the words in the fragmentary line. However, it seems extremely likely to be part of the prior line. In more ambiguous cases (especially with larger amounts of data available), it may be a better choice to omit such lines.

6 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, Pr(X = \assistance"|Y = SPAM) is calculated as follows:

$$\frac{\text{the number of times "assistance" occurs in SPAM labeled documents}}{\text{the number of words in documents labeled SPAM}}$$

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 estimate of the class conditional for a Naive Bayes Classifier.

```
In [6]: %%writefile mapper.py
        #!/usr/bin/env python
        from collections import Counter
```

```
        import itertools
        import re
        import sys

        strip_re=re.compile(r'\W+')
        split_re=re.compile(r'''(?:\s|\.|[-/:,' "@\|+*]|\\)+''')
        def tokenize(*fields):
            for field in fields:
                if field.strip() != 'NA':
                    field=re.sub(r"'ll\b", " will", field)
                    field=re.sub(r"n't\b", " not", field)
                    field=re.sub(r"'re\b", " are", field)
                    field=re.sub(r"\bit's\b", "it is", field)
                    for word in split_re.split(field.lower()):
                        word=strip_re.sub('', word)
                        if word:
                            yield word

        def main():
            if len(sys.argv) != 3:
                print >>sys.stderr, "Usage: {0} [filename] [words|*]".format(sys.argv[0])
                sys.exit(1)
            with open(sys.argv[1]) as f:
                tgts={x for x in sys.argv[2].lower().split()}
                for linenum, line in enumerate(f):
                    fields=line.split('\t')
                    if len(fields) != 4:
                        print >>sys.stderr, "Line {0}: expected 4 fields, found {1}".format(linenum, le
                        continue
                    docid=fields[0]
                    classid=('spam' if fields[1] == '1' else 'ham')
                    words=Counter(tokenize(fields[2], fields[3]))
                    print 'doc {0} {1} {2}'.format(docid, classid, sum(words.values()))
                    for word, count in words.items():
                        tgt=(word in tgts or '*' in tgts)
                        print 'term {0} {1} {2} {3} {4}'.format(docid, classid, word, count, tgt)

        if __name__ == '__main__':
            main()
```

```
Overwriting mapper.py
```

```
In [7]: %%writefile reducer.py
        #!/usr/bin/env python
        from collections import namedtuple, defaultdict, Counter
        import itertools
        import math
        import os
        import sys

        class NBClassifier(object):
            """Class to consume output of mapper.py to train a multinomial
            naive Bayes classifier, and to predict classes.
            """
            # little helpers to refer to mapper output fields by name instead of position
```

```
DocRecord=namedtuple("DocRecord",['docid', 'classid', 'count'])
TermRecord=namedtuple("TermRecord",['docid', 'classid', 'term', 'count', 'target'])

def __init__(self, alpha=0.0):
    """Create a naive bayes classifier instance with the specified smoothing:
    0 for no smoothing
    1 for Laplace smoothing
    Any other value for Lidstone smoothing
    """
    self.alpha=alpha
    self.verbose=False
    self.class_counts=Counter()
    self.doc_classes={}
    self.doc_term_counts=defaultdict(lambda:defaultdict(lambda:0))
    self.class_term_counts=defaultdict(lambda:Counter())
    self.class_tot_counts=Counter()
    self.class_priors=None
    self.cond_probs=None
    self.targets=set()

def fit(self, iterable):
    """Consume mapper records and calculate probabilities"""
    for line in iterable:
        self._add_rec(line)
    self._calc_priors()
    self._calc_cond_probs()
def predict_proba(self, doc_term_counts):
    """Return class posterior probs of a list of term counts, omitting untargeted terms"""
    posteriors={}
    used={}
    for classid, term_probs in self.cond_probs.items():
        posteriors[classid]=self.class_priors[classid]

    classid=''
    term=''
    for classid, denom in self.class_tot_counts.items():
        try:
            for term, doc_count in doc_term_counts.items():
                if term in self.targets:
                    n_terms=len(self.class_term_counts[classid])
                    term_count=self.class_term_counts[classid][term]
                    cond_prob=math.log(term_count+self.alpha)-math.log(denom+self.alpha*n_t
                    posteriors[classid]+=doc_count*cond_prob
        except ValueError:
            del(posteriors[classid])
            if self.verbose:
                print '{0} not present in class {1} term list'.format(term, classid)
            pass
    return posteriors

def predict(self, doc_term_counts):
    """Return class ID maximizing posterior prob of a list of term counts, omitting untarge
    posteriors=self.predict_proba(doc_term_counts)
    return max(posteriors, key=lambda x: posteriors[x])
```

```
    def _add_rec(self, line):
        """Consume mapper-produced doc and term records"""
        try:
            args=line.strip().split()
            if args[0]=='doc':
                if self.verbose:
                    print >>sys.stderr,"Processing doc", args[1]
                rec=NBClassifier.DocRecord(args[1], args[2], int(args[3]))
                self.doc_classes[rec.docid]=rec.classid
                self.class_counts[rec.classid]+=1
            elif args[0]=='term':
                rec=NBClassifier.TermRecord(args[1], args[2], args[3], float(args[4]), args[5])
                self.class_term_counts[rec.classid][rec.term]+=rec.count
                self.class_tot_counts[rec.classid]+=rec.count
                self.doc_term_counts[rec.docid][rec.term]+=rec.count
                if rec.target == 'True':
                    self.targets.add(rec.term)
            else:
                print >>sys.stderr, 'Unexpected row: {0}'.format(line)
        except ValueError, n:
            print >>sys.stderr, line
            raise n
    def _calc_priors(self):
        """Calculate log prior probs for a doc to be in each class"""
        doc_count=sum(self.class_counts.values())
        self.class_priors=defaultdict(lambda:0)
        for classid, count in self.class_counts.items():
            self.class_priors[classid]=math.log(1.0*count/doc_count)

    def _calc_cond_probs(self):
        self.cond_probs=defaultdict(lambda:defaultdict(lambda:0))
        for classid, denom in self.class_tot_counts.items():
            n_terms=len(self.class_term_counts[classid])
            for term, count in self.class_term_counts[classid].items():
                self.cond_probs[classid][term]=math.log(count+self.alpha)-math.log(denom+self.a

def gen_lines(files):
    """The moral equivalent of
       `itertools.chain(open(fn) for fn in files)`
    but only opening one file at a time and ensuring `close()`
    gets called appropriately
    """
    for part_file in files:
        with open(part_file) as f:
            for line in f:
                yield line
def main():
    if len(sys.argv) == 1:
        print >>sys.stderr, "Usage: {0} [filenames]*".format(sys.argv[0])
        sys.exit(1)
    # Make a classifier object
    alpha=os.environ.get('alpha',0.0)
```

```
if isinstance(alpha, basestring):
    try:
        alpha=float(alpha)
    except:
        alpha=1.0

nbc=NBClassifier(alpha=alpha)
nbc.fit(gen_lines(sys.argv[1:]))
nbc.verbose='verbose' in os.environ

print >>sys.stderr, "Processed {0} records".format(len(nbc.doc_classes))
print >>sys.stderr, "Classes:"
for classid in nbc.class_counts:
    print >>sys.stderr, "    {0: <4.4s}: {1} terms, {2} docs, {3:.3f} prior".format(
        classid,
        len(nbc.cond_probs[classid]),
        nbc.class_counts[classid],
        nbc.class_priors[classid],
    )
    if nbc.verbose:
        for term, cond in itertools.islice(sorted(nbc.cond_probs[classid].items(),key=lambda
            print >>sys.stderr, "        {0}:\t{1}".format(term, cond)

preds=[]
matches=[]
for docid, actual in nbc.doc_classes.items():
    pred=nbc.predict(nbc.doc_term_counts[docid])
    preds.append(pred)
    matches.append(pred==actual)
    print '{0}\t{1:d}\t{2:d}'.format(docid, actual=='spam', pred=='spam')
    if nbc.verbose:
        probs=nbc.predict_proba(nbc.doc_term_counts[docid])
        print >>sys.stderr, '{0: <20.20s}  pred: {1: <4.4}  match: {2:1d}  {3}'.format(
            docid, pred, matches[-1],
        ', '.join('{0}: {1:.2f}'.format(k,v) for k,v in probs.items()))
print >>sys.stderr, "Matches:",sum(matches),'of',len(matches)
print >>sys.stderr, "Accuracy:",1.0*sum(matches)/len(matches)
counts=defaultdict(lambda:0.0)
for pred, match in zip(preds, matches):
    tag=''
    if match:
        tag += 't'
    else:
        tag += 'f'
    if pred=='spam':
        tag += 'p'
    else:
        tag += 'n'
    counts[tag] += 1
for k,v in counts.items():
    print >>sys.stderr, "{0}: {1}".format(k,v)
if counts['tp']+counts['fp'] > 0:
    print >>sys.stderr, "Precision: {0:.3f}".format(
        counts['tp']/(counts['tp']+counts['fp']))
```

```
            if counts['tp']+counts['fn'] > 0:
                print >>sys.stderr, "Recall:    {0:<.3f}".format(
                    counts['tp']/(counts['tp']+counts['fn']))

        if __name__ == '__main__':
            main()
```

```
Overwriting reducer.py
```

```
In [8]: !chmod +x *.py
        !./pNaiveBayes.sh 1 bananananana
```

```
Processed 100 records
Classes:
    ham : 2724 terms, 56 docs, -0.580 prior
    spam: 3736 terms, 44 docs, -0.821 prior
Matches: 56 of 100
Accuracy: 0.56
tn: 56.0
fn: 44.0
Recall:    0.000
```

When only targeting a word not present in the dataset, we see that as expected the priors win: the classifer predicts everything to be ham.

```
In [9]: !./pNaiveBayes.sh 1 assistance
```

```
Processed 100 records
Classes:
    ham : 2724 terms, 56 docs, -0.580 prior
    spam: 3736 terms, 44 docs, -0.821 prior
Matches: 60 of 100
Accuracy: 0.6
tn: 54.0
fp: 2.0
tp: 6.0
fn: 38.0
Precision: 0.750
Recall:    0.136
```

Choosing the term 'assistance' changes 8 of these predictions to spam. This is wrong for 2 emails, but correct for 6 others.

7 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 Naive Bayes classification via the chosen list.

```
In [10]: !./pNaiveBayes.sh 1 'assistance valium enlargementWithATypo'
```

```
Processed 100 records
Classes:
    ham : 2724 terms, 56 docs, -0.580 prior
    spam: 3736 terms, 44 docs, -0.821 prior
Matches: 63 of 100
Accuracy: 0.63
tn: 54.0
fp: 2.0
tp: 9.0
fn: 35.0
Precision: 0.818
Recall:    0.205
```

Since `valium` is not present in the hams, this didn't help any. With smoothing, we see more benefit:

```
In [11]: !alpha=1 ./pNaiveBayes.sh 1 'assistance valium enlargementWithATypo'
```

```
Processed 100 records
Classes:
    ham : 2724 terms, 56 docs, -0.580 prior
    spam: 3736 terms, 44 docs, -0.821 prior
Matches: 63 of 100
Accuracy: 0.63
tn: 54.0
fp: 2.0
tp: 9.0
fn: 35.0
Precision: 0.818
Recall:    0.205
```

8 HW1.5.

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

To do so, make sure that

- mapper.py counts all occurrences of all words, and
- reducer.py performs a word-distribution-wide Naive Bayes classification.

```
In [12]: !alpha=0 ./pNaiveBayes.sh 1 '*'
```

```
Processed 100 records
Classes:
    ham : 2724 terms, 56 docs, -0.580 prior
    spam: 3736 terms, 44 docs, -0.821 prior
Matches: 100 of 100
Accuracy: 1.0
tn: 56.0
tp: 44.0
Precision: 1.000
Recall:    1.000
```

```
In [13]: !alpha=1 ./pNaiveBayes.sh 1 '*'
```

```
Processed 100 records
Classes:
    ham : 2724 terms, 56 docs, -0.580 prior
    spam: 3736 terms, 44 docs, -0.821 prior
Matches: 100 of 100
Accuracy: 1.0
tn: 56.0
tp: 44.0
Precision: 1.000
Recall:    1.000
```

Comments: Even the tiniest bit of smoothing appears to be required to avoid blowing up by trying to take the log of zero. Once that is added, the classifier predicts correctly for all the training data.

Addendum: I modified the classifier to reject classes that trigger log(0) thus:

```
del posteriors[classid]
```

After which, the unsmoothed case performs perfectly as well. Running with verbose flag enabled confirms there is always at least one word present in each document that is present in only the ham or the spam term lists.

In all cases, mapper.py will read in a portion of the email data, count some words and print out counts to a file.

While the utility of the reducer will change significantly across steps, it will always be responsible for reading in counts of words and collating data.

In all cases you should apply a Laplace (add-1) smoothing to the classifier (always on the reducer side) to safeguard code against low-data.

You will find in the starter code (pNaiveBayes.sh) that the basic operations (e.g., splitting the original data, scheduling the mappers, waiting, running the reducer, and cleaning up the intermediate data files) are taken care of, and that the portion of this assignment left for you is in python and will involve regular expressions, counting with objects, and some light math.

For a quick reference on the construction of the classifier that you will code, please consult the "Document Classification" section of the following wikipedia page:

https://en.wikipedia.org/wiki/Naive_Bayes_classifier#Document_classification

the original paper by our curators of the Enron email data:

http://www.aueb.gr/users/ion/docs/ceas2006_paper.pdf

or the recording of this week's live lecture that you will find on the LMS.

9 HW1.6.

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

It always a good idea to test your solutions against publicly available libraries such as SciKit-Learn, The Machine Learning toolkit available in Python. In this exercise, we benchmark ourselves against the SciKit-Learn implementation of multinomial Naive Bayes. For more information on this implementation see: http://scikit-learn.org/stable/modules/naive_bayes.html more

Let's define Training error = misclassification rate with respect to a training set. It is more formally defined here:

Let DF represent the training set in the following:

$$\text{Err}(\text{Model}, DF) = \frac{|\{(X, c(X)) \in DF : c(X) \neq \text{Model}(x)\}|}{|DF|}$$

Where $||$ denotes set cardinality; $c(X)$ denotes the class of the tuple X in DF; and Model(X) denotes the class inferred by the model Model

In this exercise, please complete the following:

1. Run the Multinomial Naive Bayes algorithm (using default settings) from SciKit-Learn over the same training data used in HW1.5 and report the Training error (please note some data preparation might be needed to get the Multinomial Naive Bayes algorithm from SkiKit-Learn to run over this dataset)
2. Run the Bernoulli Naive Bayes algorithm from SciKit-Learn (using default settings) over the same training data used in HW1.5 and report the Training error
3. Run the Multinomial Naive Bayes algorithm you developed for HW1.5 over the same data used HW1.5 and report the Training error
4. Please prepare a table to present your results
5. Explain/justify any differences in terms of training error rates over the dataset in HW1.5 between your Multinomial Naive Bayes implementation (in Map Reduce) versus the Multinomial Naive Bayes implementation in SciKit-Learn (Hint: smoothing, which we will discuss in next lecture)
6. Discuss the performance differences in terms of training error rates over the dataset in HW1.5 between the Multinomial Naive Bayes implementation in SciKit-Learn with the Bernoulli Naive Bayes implementation in SciKit-Learn

In [14]:
```
# 1.6.1: Quick and easy ingest using pandas
import numpy as np
import pandas as pd

def read_data(fn):
    data=pd.read_csv(fn, sep='\t', header=None, na_values=['NA'])
    data.columns=['id','spam', 'subject', 'body']
    data.index=data['id']
    data = data.replace(np.nan,'')
    data['doc']=data['subject']+' '+data['body']
    return data[['spam','doc']]
```

In [15]:
```
data=read_data("enronemail_1h.txt")
data.head(10)
```

Out[15]:

id	spam	doc
0001.1999-12-10.farmer	0	christmas tree farm pictures
0001.1999-12-10.kaminski	0	re: rankings thank you.
0001.2000-01-17.beck	0	leadership development pilot sally: what ti...
0001.2000-06-06.lokay	0	key dates and impact of upcoming sap implement...
0001.2001-02-07.kitchen	0	key hr issues going forward a) year end revi...
0001.2001-04-02.williams	0	re: quasi good morning, i'd love to go get ...
0002.1999-12-13.farmer	0	vastar resources, inc. gary, production from...
0002.2001-02-07.kitchen	0	congrats! contratulations on the execution o...
0002.2001-05-25.SA_and_HP	1	fw: this is the solution i mentioned lsc oo ...
0002.2003-12-18.GP	1	adv: space saving computer to replace that bi...

In [16]:
```
#1.6.1 copied to use the same tokenizer between sklearn and hand-written NB classifiers
import re

strip_re=re.compile(r'\W+')
split_re=re.compile(r'''(?:\s|\.|[-/:,' "@\|+*]|\\)+''')
def tokenize(*fields):
    for field in fields:
        if field.strip() != 'NA':
            field=re.sub(r"'ll\b", " will", field)
            field=re.sub(r"n't\b", " not", field)
            field=re.sub(r"'re\b", " are", field)
            field=re.sub(r"\bit's\b", "it is", field)
            for word in split_re.split(field.lower()):
                word=strip_re.sub('', word)
                if word:
                    yield word
```

In [17]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import classification_report, accuracy_score
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.pipeline import Pipeline

def hw16_1(data, alpha):
    cv=CountVectorizer(tokenizer=tokenize)
    mnb=MultinomialNB(alpha=alpha)
    pl=Pipeline([('cv',cv),('mnb',mnb)])
    pl.fit(data['doc'],data['spam'])
    pred=pl.predict(data['doc'])
    print "alpha={0:.3f}".format(alpha)
    print "Training_error={0:.3f}".format(1-accuracy_score(data['spam'], pred))
    print classification_report(data['spam'], pred, target_names=['ham', 'spam'])
```

In [18]:
```
hw16_1(read_data("enronemail_1h.txt"),0)
hw16_1(read_data("enronemail_1h.txt"),1)
```

```
alpha=0.000
Training_error=0.000
             precision    recall  f1-score   support

        ham       1.00      1.00      1.00        56
       spam       1.00      1.00      1.00        44

avg / total       1.00      1.00      1.00       100

alpha=1.000
Training_error=0.000
             precision    recall  f1-score   support

        ham       1.00      1.00      1.00        56
       spam       1.00      1.00      1.00        44

avg / total       1.00      1.00      1.00       100
```

In [19]:
```
def hw16_2(data, alpha):
    cv=CountVectorizer(tokenizer=tokenize)
```

```
    bnb=BernoulliNB(alpha=alpha,binarize=1)
    pl=Pipeline([('cv',cv),('bnb',bnb)])
    pl.fit(data['doc'],data['spam'])
    pred=pl.predict(data['doc'])
    print "alpha={0:.3f}".format(alpha)
    print "Training_error={0:.3f}".format(1-accuracy_score(data['spam'], pred))
    print classification_report(data['spam'], pred, target_names=['ham', 'spam'])
```

```
In [20]: hw16_2(read_data("enronemail_1h.txt"),0.0)
         hw16_2(read_data("enronemail_1h.txt"),0.5)
         hw16_2(read_data("enronemail_1h.txt"),1.0)
```

```
alpha=0.000
Training_error=0.060
             precision    recall  f1-score   support

        ham       0.90      1.00      0.95        56
       spam       1.00      0.86      0.93        44

avg / total       0.95      0.94      0.94       100

alpha=0.500
Training_error=0.290
             precision    recall  f1-score   support

        ham       0.66      1.00      0.79        56
       spam       1.00      0.34      0.51        44

avg / total       0.81      0.71      0.67       100

alpha=1.000
Training_error=0.340
             precision    recall  f1-score   support

        ham       0.62      1.00      0.77        56
       spam       1.00      0.23      0.37        44

avg / total       0.79      0.66      0.59       100
```

9.1 1.6.3

Training error as defined above is $(1 - \text{Accuracy}(\text{Model}, DF))$; so the classifier in problem 1.5 has a training error of 0 or 1, depending on whether smoothing is enabled or not, per the accuracy figure logged.

9.2 1.6.4 Training error table

	Model	α	Training error
1	sklearn Multinomial NB	0.00	0.00
2	sklearn Multinomial NB	1.00	0.00
3	sklearn Bernoulli NB	0.00	0.06
4	sklearn Bernoulli NB	0.50	0.29
5	sklearn Bernoulli NB	1.00	0.34
6	Prob 1.5 Multinomial NB	0.00	0.00
7	Prob 1.5 Multinomial NB	1.00	0.00

9.3 1.6.5 Discussion

The main oddity encountered was how the problem 1.5 classifier exploded without smoothing. I would not have expected every mail to include a term present exclusively in the ham or spam term lists, but that seems to be the only reason every classification would fail without smoothing. After altering details of how the invalid operation is avoided, the classifer correctly classifies each document (reflected above).

When weighting is enabled, rather than outright rejecting terms not present in a class, the class conditional (log) probability should take the form:

$$d \log(\alpha) - d \log\left(\alpha \cdot |T_C| + \sum_{t \in T_C} n_t\right)$$

where T_C is the set of terms in the class, n_t is the total number of times term t occurs in the class, and d is the number of times the out-of-vocabulary term occurs in the document. This corresponds to a term that appears α times in the class, so smoothing values between 0 to 1 make missing terms behave like they are present but less frequent than any other term in the vocabulary.

In [ ]: