In [1]:
```
%pylab inline
import pylab as pl
import numpy as np

# Some nice default configuration for plots
pl.rcParams['figure.figsize'] = 10, 7.5
pl.rcParams['axes.grid'] = True
pl.gray()
```

```
Welcome to pylab, a matplotlib-based Python environment [backend:
module://IPython.zmq.pylab.backend_inline].
For more information, type 'help(pylab)'.
```

Text Feature Extraction for Classification and Clustering

Outline of this section:

- Turn a corpus of text documents into **feature vectors** using a **Bag of Words** representation,
- Train a simple text classifier on the feature vectors,
- Wrap the vectorizer and the classifier with a **pipeline**,
- Cross-validation and **model selection** on the pipeline.

Text Classification in 20 lines of Python

Let's start by implementing a canonical text classification example:

- The 20 newsgroups dataset: around 18000 text posts from 20 newsgroups forums
- Bag of Words features extraction with TF-IDF weighting
- Naive Bayes classifier or Linear Support Vector Machine for the classifier itself

In [2]:
```
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB

# Load the text data
categories = [
    'alt.atheism',
    'talk.religion.misc',
    'comp.graphics',
    'sci.space',
]
twenty_train_small = load_files('../datasets/20news-bydate-train/',
    categories=categories, charset='latin-1')
twenty_test_small = load_files('../datasets/20news-bydate-test/',
    categories=categories, charset='latin-1')

# Turn the text documents into vectors of word frequencies
vectorizer = TfidfVectorizer(min_df=2)
```

```python
X_train = vectorizer.fit_transform(twenty_train_small.data)
y_train = twenty_train_small.target

# Fit a classifier on the training set
classifier = MultinomialNB().fit(X_train, y_train)
print("Training score: {0:.1f}%".format(
    classifier.score(X_train, y_train) * 100))

# Evaluate the classifier on the testing set
X_test = vectorizer.transform(twenty_test_small.data)
y_test = twenty_test_small.target
print("Testing score: {0:.1f}%".format(
    classifier.score(X_test, y_test) * 100))
```

```
Training score: 95.1%
Testing score: 85.1%
```

Here is a workflow diagram summary of what happened previously:

Let's now decompose what we just did to understand and customize each step.

Loading the Dataset

Let's explore the dataset loading utility without passing a list of categories: in this case we load the full 20 newsgroups dataset in memory. The source website for the 20 newsgroups already provides a date-based train / test split that is made available using the `subset` keyword argument:

In [3]:
```
ls -l ../datasets/
```

```
total 28256
```

```
drwxr-xr-x  22 alexwchen  staff       748 18 Mar  2003 20news-bydate-test/
drwxr-xr-x  22 alexwchen  staff       748 18 Mar  2003 20news-bydate-train/
-rw-r--r--   1 alexwchen  staff  14464277  7 Jun 16:36 20news-bydate.tar.gz
```

In [4]:
```
ls -lh ../datasets/20news-bydate-train
```

```
total 0
drwxr-xr-x  482 alexwchen  staff    16K 18 Mar  2003 alt.atheism/
drwxr-xr-x  586 alexwchen  staff    19K 18 Mar  2003 comp.graphics/
drwxr-xr-x  593 alexwchen  staff    20K 18 Mar  2003 comp.os.ms-windows.misc/
drwxr-xr-x  592 alexwchen  staff    20K 18 Mar  2003 comp.sys.ibm.pc.hardware/
drwxr-xr-x  580 alexwchen  staff    19K 18 Mar  2003 comp.sys.mac.hardware/
drwxr-xr-x  595 alexwchen  staff    20K 18 Mar  2003 comp.windows.x/
drwxr-xr-x  587 alexwchen  staff    19K 18 Mar  2003 misc.forsale/
drwxr-xr-x  596 alexwchen  staff    20K 18 Mar  2003 rec.autos/
drwxr-xr-x  600 alexwchen  staff    20K 18 Mar  2003 rec.motorcycles/
drwxr-xr-x  599 alexwchen  staff    20K 18 Mar  2003 rec.sport.baseball/
drwxr-xr-x  602 alexwchen  staff    20K 18 Mar  2003 rec.sport.hockey/
drwxr-xr-x  597 alexwchen  staff    20K 18 Mar  2003 sci.crypt/
drwxr-xr-x  593 alexwchen  staff    20K 18 Mar  2003 sci.electronics/
drwxr-xr-x  596 alexwchen  staff    20K 18 Mar  2003 sci.med/
drwxr-xr-x  595 alexwchen  staff    20K 18 Mar  2003 sci.space/
drwxr-xr-x  601 alexwchen  staff    20K 18 Mar  2003 soc.religion.christian/
drwxr-xr-x  548 alexwchen  staff    18K 18 Mar  2003 talk.politics.guns/
drwxr-xr-x  566 alexwchen  staff    19K 18 Mar  2003 talk.politics.mideast/
drwxr-xr-x  467 alexwchen  staff    16K 18 Mar  2003 talk.politics.misc/
drwxr-xr-x  379 alexwchen  staff    13K 18 Mar  2003 talk.religion.misc/
```

In [5]:
```
ls -lh ../datasets/20news-bydate-train/alt.atheism/
```

```
total 4480
-rw-r--r--  1 alexwchen  staff    12K 18 Mar  2003 49960
-rw-r--r--  1 alexwchen  staff    31K 18 Mar  2003 51060
-rw-r--r--  1 alexwchen  staff   4.0K 18 Mar  2003 51119
-rw-r--r--  1 alexwchen  staff   1.6K 18 Mar  2003 51120
-rw-r--r--  1 alexwchen  staff   773B 18 Mar  2003 51121
-rw-r--r--  1 alexwchen  staff   4.8K 18 Mar  2003 51122
-rw-r--r--  1 alexwchen  staff   618B 18 Mar  2003 51123
-rw-r--r--  1 alexwchen  staff   1.4K 18 Mar  2003 51124
-rw-r--r--  1 alexwchen  staff   2.7K 18 Mar  2003 51125
-rw-r--r--  1 alexwchen  staff   427B 18 Mar  2003 51126
-rw-r--r--  1 alexwchen  staff   742B 18 Mar  2003 51127
-rw-r--r--  1 alexwchen  staff   650B 18 Mar  2003 51128
-rw-r--r--  1 alexwchen  staff   1.3K 18 Mar  2003 51130
-rw-r--r--  1 alexwchen  staff   2.3K 18 Mar  2003 51131
-rw-r--r--  1 alexwchen  staff   2.6K 18 Mar  2003 51132
-rw-r--r--  1 alexwchen  staff   1.5K 18 Mar  2003 51133
-rw-r--r--  1 alexwchen  staff   1.2K 18 Mar  2003 51134
-rw-r--r--  1 alexwchen  staff   1.6K 18 Mar  2003 51135
-rw-r--r--  1 alexwchen  staff   2.1K 18 Mar  2003 51136
```

The `load_files` function can load text files from a 2 levels folder structure assuming folder names represent categories:

In [ ]:
```
#print(load_files.__doc__)
```

In [6]:
```
all_twenty_train = load_files('../datasets/20news-bydate-train/',
  charset='latin-1', random_state=42)
all_twenty_test = load_files('../datasets/20news-bydate-test/',
```

```
    charset='latin-1', random_state=42)
```

In [7]:
```
all_target_names = all_twenty_train.target_names
all_target_names
```

Out[7]:
```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

In [8]:
```
all_twenty_train.target
```

Out[8]:
```
array([12,  6,  9, ...,  9,  1, 12])
```

In [9]:
```
all_twenty_train.target.shape
```

Out[9]:
```
(11314,)
```

In [10]:
```
all_twenty_test.target.shape
```

Out[10]:
```
(7532,)
```

In [11]:
```
len(all_twenty_train.data)
```

Out[11]:
```
11314
```

In [12]:
```
type(all_twenty_train.data[0])
```

Out[12]:
```
unicode
```

In [13]:
```
def display_sample(i, dataset):
    print("Class name: " + dataset.target_names[dataset.target[i]])
    print("Text content:\n")
    print(dataset.data[i])
```

In [14]:
```
display_sample(0, all_twenty_train)
```

```
Class name: sci.electronics
Text content:
```

```
From: wtm@uhura.neoucom.edu (Bill Mayhew)
Subject: Re: How to the disks copy protected.
Organization: Northeastern Ohio Universities College of Medicine
Lines: 23

Write a good manual to go with the software.  The hassle of
photocopying the manual is offset by simplicity of purchasing
the package for only $15.  Also, consider offering an inexpensive
but attractive perc for registered users.  For instance, a coffee
mug.  You could produce and mail the incentive for a couple of
dollars, so consider pricing the product at $17.95.

You're lucky if only 20% of the instances of your program in use
are non-licensed users.

The best approach is to estimate your loss and accomodate that into
your price structure.  Sure it hurts legitimate users, but too bad.
Retailers have to charge off loss to shoplifters onto paying
customers; the software industry is the same.

Unless your product is exceptionally unique, using an ostensibly
copy-proof disk will just send your customers to the competetion.


--
Bill Mayhew      NEOUCOM Computer Services Department
Rootstown, OH  44272-9995  USA    phone: 216-325-2511
wtm@uhura.neoucom.edu (140.220.1.1)    146.580: N8WED
```

In [15]:
```
display_sample(1, all_twenty_train)
```

```
Class name: misc.forsale
Text content:

From: andy@SAIL.Stanford.EDU (Andy Freeman)
Subject: Re: Catalog of Hard-to-Find PC Enhancements (Repost)
Organization: Computer Science Department,  Stanford University.
Lines: 33

>andy@SAIL.Stanford.EDU (Andy Freeman) writes:
>> >In article <C5ELME.4z4@unix.portal.com> jdoll@shell.portal.com (Joe Doll) wr
>> >>   "The Catalog of Personal Computing Tools for Engineers and Scien-
>> >>   tists" lists hardware cards and application software packages for
>> >>   PC/XT/AT/PS/2 class machines.  Focus is on engineering and scien-
>> >>   tific applications of PCs, such as data acquisition/control,
>> >>   design automation, and data analysis and presentation.
>> >
>> >>   If you would like a free copy, reply with your (U. S. Postal)
>> >>   mailing address.
>>
>> Don't bother - it never comes.  It's a cheap trick for building a
>> mailing list to sell if my junk mail flow is any indication.
>>
>> -andy sent his address months ago
>
>Perhaps we can get Portal to nuke this weasal.  I never received a
>catalog either.  If that person doesn't respond to a growing flame, then
>we can assume that we'yall look forward to lotsa junk mail.
```

```
I don't want him nuked, I want him to be honest.  The junk mail has
been much more interesting than the promised catalog.  If I'd known
what I was going to get, I wouldn't have hesitated.  I wouldn't be
surprised if there were other folks who looked at the ad and said
"nope" but who would be very interested in the junk mail that results.
Similarly, there are people who wanted the advertised catalog who
aren't happy with the junk they got instead.

The folks buying the mailing lists would prefer an honest ad, and
so would the people reading it.

-andy
--
```

Let's compute the (uncompressed, in-memory) size of the training and test sets in MB assuming an 8 bit encoding (in this case, all chars can be encoded using the latin-1 charset).

In [16]:
```
def text_size(text, charset='iso-8859-1'):
    return len(text.encode(charset)) * 8 * 1e-6

train_size_mb = sum(text_size(text) for text in all_twenty_train.data)
test_size_mb = sum(text_size(text) for text in all_twenty_test.data)

print("Training set size: {0} MB".format(int(train_size_mb)))
print("Testing set size: {0} MB".format(int(test_size_mb)))
```

```
Training set size: 176 MB
Testing set size: 110 MB
```

If we only consider a small subset of the 4 categories selected from the initial example:

In [17]:
```
train_small_size_mb = sum(text_size(text) for text in twenty_train_small.data)
test_small_size_mb = sum(text_size(text) for text in twenty_test_small.data)

print("Training set size: {0} MB".format(int(train_small_size_mb)))
print("Testing set size: {0} MB".format(int(test_small_size_mb)))
```

```
Training set size: 31 MB
Testing set size: 22 MB
```

Extracting Text Features

In [18]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

TfidfVectorizer()
```

Out[18]:
```
TfidfVectorizer(analyzer='word', binary=False, charset='utf-8',
        charset_error='strict', dtype=<type 'long'>, input='content',
        lowercase=True, max_df=1.0, max_features=None, max_n=None,
        min_df=2, min_n=None, ngram_range=(1, 1), norm='l2',
        preprocessor=None, smooth_idf=True, stop_words=None,
        strip_accents=None, sublinear_tf=False,
        token_pattern=u'(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

In [21]:
```
vectorizer = TfidfVectorizer(min_df=1)

%time X_train_small = vectorizer.fit_transform(twenty_train_small.data)
```

```
CPU times: user 2.76 s, sys: 0.07 s, total: 2.83 s
Wall time: 3.09 s
```

The results is not a `numpy.array` but instead a `scipy.sparse` matrix. This datastructure is quite similar to a 2D numpy array but it does not store the zeros.

In [22]:
```
X_train_small
```

Out[22]:
```
<2034x34118 sparse matrix of type '<type 'numpy.float64'>'
	with 323433 stored elements in Compressed Sparse Row format>
```

scipy.sparse matrices also have a shape attribute to access the dimensions:

In [23]:
```
n_samples, n_features = X_train_small.shape
```

This dataset has around 2000 samples (the rows of the data matrix):

In [24]:
```
n_samples
```

Out[24]:
```
2034
```

This is the same value as the number of strings in the original list of text documents:

In [25]:
```
len(twenty_train_small.data)
```

Out[25]:
```
2034
```

The columns represent the individual token occurrences:

In [26]:
```
n_features
```

Out[26]:
```
34118
```

This number is the size of the vocabulary of the model extracted during fit in a Python dictionary:

In [27]:
```
type(vectorizer.vocabulary_)
```

Out[27]:
```
dict
```

In [28]:
```
len(vectorizer.vocabulary_)
```

Out[28]:
```
34118
```

The keys of the `vocabulary_` attribute are also called feature names and can be accessed as a list of strings.

In [29]:
```
len(vectorizer.get_feature_names())
```

Out[29]:
```
34118
```

Here are the first 10 elements (sorted in lexicographical order):

In [30]:
```
vectorizer.get_feature_names()[:10]
```

Out[30]:
```
[u'00',
 u'000',
 u'0000',
 u'00000',
 u'000000',
 u'000005102000',
 u'000021',
 u'000062david42',
 u'0000vec',
 u'0001']
```

Let's have a look at the features from the middle:

In [31]:
```
vectorizer.get_feature_names()[n_features / 2:n_features / 2 + 10]
```

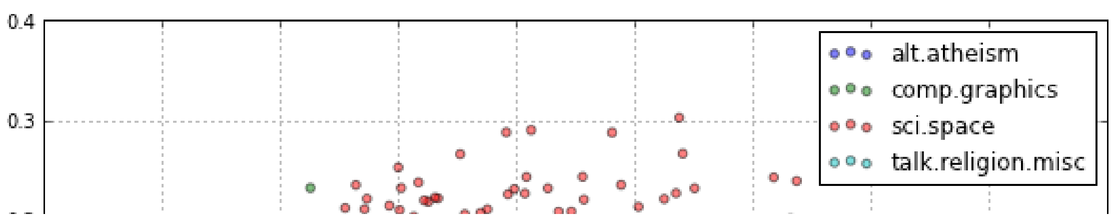
Out[31]:
```
[u'inadequate',
 u'inala',
 u'inalienable',
 u'inane',
 u'inanimate',
 u'inapplicable',
 u'inappropriate',
 u'inappropriately',
 u'inaudible',
 u'inbreeding']
```

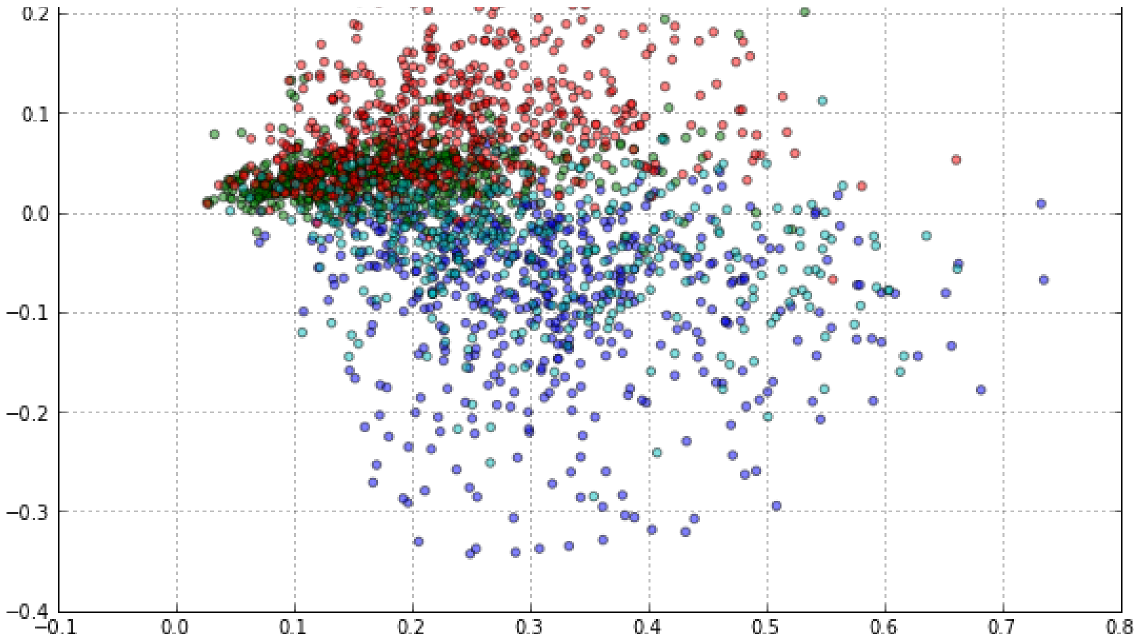
Now that we have extracted a vector representation of the data, it's a good idea to project the data on the first 2D of a Principal Component Analysis to get a feel of the data. Note that the `RandomizedPCA` class can accept `scipy.sparse` matrices as input (as an alternative to numpy arrays):

In [32]:
```
from sklearn.decomposition import RandomizedPCA

%time X_train_small_pca = RandomizedPCA(n_components=2).fit_transform(X_train_small)
```

```
CPU times: user 0.20 s, sys: 0.03 s, total: 0.22 s
Wall time: 0.61 s
```

In [33]:
```
from itertools import cycle

colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
for i, c in zip(np.unique(y_train), cycle(colors)):
    pl.scatter(X_train_small_pca[y_train == i, 0],
               X_train_small_pca[y_train == i, 1],
               c=c, label=twenty_train_small.target_names[i], alpha=0.5)

_ = pl.legend(loc='best')
```





We can observe that there is a large overlap of the samples from different categories. This is to be expected as the PCA linear projection projects data from a 34118 dimensional space down to 2 dimensions: data that is linearly separable in 34118D is often no longer linearly separable in 2D.

Still we can notice an interesting pattern: the newsgroups on religion and atheism occupy the much the same region and computer graphics and space science / space overlap more together than they do with the religion or atheism newsgroups.

Training a Classifier on Text Features

We have previously extracted a vector representation of the training corpus and put it into a variable name `X_train_small`. To train a supervised model, in this case a classifier, we also need

In [34]:
```
y_train_small = twenty_train_small.target
```

In [35]:
```
y_train_small.shape
```

Out[35]:
```
(2034,)
```

In [36]:
```
y_train_small
```

Out[36]:
```
array([1, 2, 2, ..., 2, 1, 1])
```

We can shape that we have the same number of samples for the input data and the labels:

In [37]:
```
X_train_small.shape[0] == y_train_small.shape[0]
```

Out[37]:
```
True
```

We can now train a classifier, for instance a Multinomial Naive Bayesian classifier:

In [38]:
```
from sklearn.naive_bayes import MultinomialNB
```

```
clf = MultinomialNB(alpha=0.1)
clf
```

Out[38]:
```
MultinomialNB(alpha=0.1, class_prior=None, fit_prior=True)
```

In [39]:
```
clf.fit(X_train_small, y_train_small)
```

Out[39]:
```
MultinomialNB(alpha=0.1, class_prior=None, fit_prior=True)
```

We can now evaluate the classifier on the testing set. Let's first use the builtin score function, which is the rate of correct classification in the test set:

In [40]:
```
X_test_small = vectorizer.transform(twenty_test_small.data)
y_test_small = twenty_test_small.target
```

In [41]:
```
X_test_small.shape
```

Out[41]:
```
(1353, 34118)
```

In [42]:
```
y_test_small.shape
```

Out[42]:
```
(1353,)
```

In [43]:
```
clf.score(X_test_small, y_test_small)
```

Out[43]:
```
0.89652623798965259
```

We can also compute the score on the test set and observe that the model is both overfitting and underfitting a bit at the same time:

In [44]:
```
clf.score(X_train_small, y_train_small)
```

Out[44]:
```
0.99262536873156337
```

Introspecting the Behavior of the Text Vectorizer

The text vectorizer has many parameters to customize it's behavior, in particular how it extracts tokens:

In [45]:
```
TfidfVectorizer()
```

Out[45]:
```
TfidfVectorizer(analyzer='word', binary=False, charset='utf-8',
        charset_error='strict', dtype=<type 'long'>, input='content',
        lowercase=True, max_df=1.0, max_features=None, max_n=None,
        min_df=2, min_n=None, ngram_range=(1, 1), norm='l2',
        preprocessor=None, smooth_idf=True, stop_words=None,
        strip_accents=None, sublinear_tf=False,
        token_pattern=u'(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

In [46]:
```
print(TfidfVectorizer.__doc__)
```

```
Convert a collection of raw documents to a matrix of TF-IDF features.
```

```
    Convert a collection of raw documents to a matrix of TF-IDF features.

        Equivalent to CountVectorizer followed by TfidfTransformer.

        Parameters
        ----------
        input : string {'filename', 'file', 'content'}
            If filename, the sequence passed as an argument to fit is
            expected to be a list of filenames that need reading to fetch
            the raw content to analyze.

            If 'file', the sequence items must have 'read' method (file-like
            object) it is called to fetch the bytes in memory.

            Otherwise the input is expected to be the sequence strings or
            bytes items are expected to be analyzed directly.

        charset : string, 'utf-8' by default.
            If bytes or files are given to analyze, this charset is used to
            decode.
```

The easiest way to introspect what the vectorizer is actually doing for a given test of parameters is call the `vectorizer.build_analyzer()` to get an instance of the text analyzer it uses to process the text:

In [47]:
```
analyzer = TfidfVectorizer().build_analyzer()
analyzer("I love scikit-learn: this is a cool Python lib!")
```

Out[47]:
```
[u'love', u'scikit', u'learn', u'this', u'is', u'cool', u'python', u'lib']
```

You can notice that all the tokens are lowercase, that the single letter word "I" was dropped, and that hyphenation is used. Let's change some of that default behavior:

In [48]:
```
analyzer = TfidfVectorizer(
    preprocessor=lambda text: text,  # disable lowercasing
    token_pattern=ur'(?u)\b[\w-]+\b', # treat hyphen as a letter
                                      # do not exclude single letter tokens
).build_analyzer()

analyzer("I love scikit-learn: this is a cool Python lib!")
```

Out[48]:
```
[u'I',
 u'love',
 u'scikit-learn',
 u'this',
 u'is',
 u'a',
 u'cool',
 u'Python',
 u'lib']
```

The analyzer name comes from the Lucene parlance: it wraps the sequential application of:

- text preprocessing (processing the text documents as a whole, e.g. lowercasing)
- text tokenization (splitting the document into a sequence of tokens)
- token filtering and recombination (e.g. n-grams extraction, see later)

The analyzer system of scikit-learn is much more basic than lucene's though.

Exercise:

- Write a pre-processor callable (e.g. a python function) to remove the headers of the text a newsgroup post.
- Vectorize the data again and measure the impact on performance of removing the header info from the dataset.
- Do you expect the performance of the model to improve or decrease? What is the score of a uniform random classifier on the same dataset?

Hint: the `TfidfVectorizer` class can accept python functions to customize the `preprocessor`, `tokenizer` or `analyzer` stages of the vectorizer.

- type `TfidfVectorizer()` alone in a cell to see the default value of the parameters
- type `TfidfVectorizer.__doc__` to print the constructor parameters doc or `?` suffix operator on a any Python class or method to read the docstring or even the `??` operator to read the source code.

Solution:

Let's write a Python function to strip the post headers and only retain the body (text after the first blank line):

In [49]:
```
def strip_headers(post):
    """Find the first blank line and drop the headers to keep the body"""
    if '\n\n' in post:
        headers, body = post.split('\n\n', 1)
        return body.lower()
    else:
        # Unexpected post inner-structure, be conservative
        # and keep everything
        return post.lower()
```

Let's try it on the first post. Here is the original post content, including the headers:

In [50]:
```
original_text = all_twenty_train.data[0]
print(original_text)
```

```
From: wtm@uhura.neoucom.edu (Bill Mayhew)
Subject: Re: How to the disks copy protected.
Organization: Northeastern Ohio Universities College of Medicine
Lines: 23

Write a good manual to go with the software.  The hassle of
photocopying the manual is offset by simplicity of purchasing
the package for only $15.  Also, consider offering an inexpensive
but attractive perc for registered users.  For instance, a coffee
mug.  You could produce and mail the incentive for a couple of
dollars, so consider pricing the product at $17.95.

You're lucky if only 20% of the instances of your program in use
are non-licensed users.

The best approach is to estimate your loss and accomodate that into
your price structure.  Sure it hurts legitimate users, but too bad.
Retailers have to charge off loss to shoplifters onto paying
customers; the software industry is the same.

Unless your product is exceptionally unique, using an ostensibly
copy-proof disk will just send your customers to the competetion.
```

```
--
Bill Mayhew      NEOUCOM Computer Services Department
Rootstown, OH  44272-9995  USA    phone: 216-325-2511
wtm@uhura.neoucom.edu (140.220.1.1)    146.580: N8WED
```

Here is the result of applying our header stripping function:

In [51]:
```
text_body = strip_headers(original_text)
print(text_body)
```

```
write a good manual to go with the software.  the hassle of
photocopying the manual is offset by simplicity of purchasing
the package for only $15.  also, consider offering an inexpensive
but attractive perc for registered users.  for instance, a coffee
mug.  you could produce and mail the incentive for a couple of
dollars, so consider pricing the product at $17.95.

you're lucky if only 20% of the instances of your program in use
are non-licensed users.

the best approach is to estimate your loss and accomodate that into
your price structure.  sure it hurts legitimate users, but too bad.
retailers have to charge off loss to shoplifters onto paying
customers; the software industry is the same.

unless your product is exceptionally unique, using an ostensibly
copy-proof disk will just send your customers to the competetion.


--
bill mayhew      neoucom computer services department
rootstown, oh  44272-9995  usa    phone: 216-325-2511
wtm@uhura.neoucom.edu (140.220.1.1)    146.580: n8wed
```

Let's plug our function in the vectorizer and retrain a naive Bayes classifier (as done initially):

In [52]:
```
strip_vectorizer = TfidfVectorizer(preprocessor=strip_headers, min_df=2)
X_train_small_stripped = strip_vectorizer.fit_transform(
    twenty_train_small.data)

y_train_small_stripped = twenty_train_small.target

classifier = MultinomialNB().fit(
  X_train_small_stripped, y_train_small_stripped)

print("Training score: {0:.1f}%".format(
    classifier.score(X_train_small_stripped, y_train_small_stripped) * 100))

X_test_small_stripped = strip_vectorizer.transform(twenty_test_small.data)
y_test_small_stripped = twenty_test_small.target
print("Testing score: {0:.1f}%".format(
    classifier.score(X_test_small_stripped, y_test_small_stripped) * 100))
```

```
Training score: 93.3%
Testing score: 82.2%
```

So indeed the header data is making the problem easier (cheating one could say) but naive Bayes classifier can still guess 80% of the time against 1 / 4 == 25% mean score for a random guessing on the small subset with 4 target categories.

Model Selection of the Naive Bayes Classifier Parameter Alone

The `MultinomialNB` class is a good baseline classifier for text as it's fast and has few parameters to tweak:

In [53]:
```
MultinomialNB()
```

Out[53]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [54]:
```
print(MultinomialNB.__doc__)
```

```
Naive Bayes classifier for multinomial models

The multinomial Naive Bayes classifier is suitable for classification with
discrete features (e.g., word counts for text classification). The
multinomial distribution normally requires integer feature counts. However,
in practice, fractional counts such as tf-idf may also work.

Parameters
----------
alpha : float, optional (default=1.0)
    Additive (Laplace/Lidstone) smoothing parameter
    (0 for no smoothing).

fit_prior : boolean
    Whether to learn class prior probabilities or not.
    If false, a uniform prior will be used.

class_prior : array-like, size=[n_classes,]
    Prior probabilities of the classes. If specified the priors are not
    adjusted according to the data.

Attributes
----------
`intercept_`, `class_log_prior_` : array, shape = [n_classes]
    Smoothed empirical log probability for each class.

`feature_log_prob_`, `coef_` : array, shape = [n_classes, n_features]
    Empirical log probability of features
    given a class, P(x_i|y).

    (`intercept_` and `coef_` are properties
    referring to `class_log_prior_` and
    `feature_log_prob_`, respectively.)

Examples
--------
>>> import numpy as np
>>> X = np.random.randint(5, size=(6, 100))
>>> Y = np.array([1, 2, 3, 4, 5, 6])
>>> from sklearn.naive_bayes import MultinomialNB
```

```
    >>> clf = MultinomialNB()
    >>> clf.fit(X, Y)
    MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
    >>> print(clf.predict(X[2]))
    [3]

    Notes
    -----
    For the rationale behind the names `coef_` and `intercept_`, i.e.
    naive Bayes as a linear classifier, see J. Rennie et al. (2003),
    Tackling the poor assumptions of naive Bayes text classifiers, ICML.
```

By reading the doc we can see that the `alpha` parameter is a good candidate to adjust the model for the bias (underfitting) vs variance (overfitting) trade-off.

Exercise:

- use the `sklearn.grid_search.GridSearchCV` or the `model_selection.RandomizedGridSeach` utility function from the previous chapters to find a good value for the parameter `alpha`
- plots the validation scores (and optionally the training scores) for each value of alpha and identify the areas where model overfitts or underfitts.

Hints:

- you can search for values of alpha in the range [0.00001 - 1] using a logarithmic scale
- `RandomizedGridSearch` also has a `launch_for_arrays` method as an alternative to `launch_for_splits` in case the CV splits have not been precomputed in advance. 1

Solution:

In [55]:
```
from IPython.parallel import Client

client = Client()
len(client)
```

Out[55]:
```
2
```

In [56]:
```
import sys
if not '..' in sys.path: sys.path.append('..')

import numpy as np
import model_selection, mmap_utils
reload(model_selection), reload(mmap_utils)


nb_search = model_selection.RandomizedGridSeach(client.load_balanced_view())
```

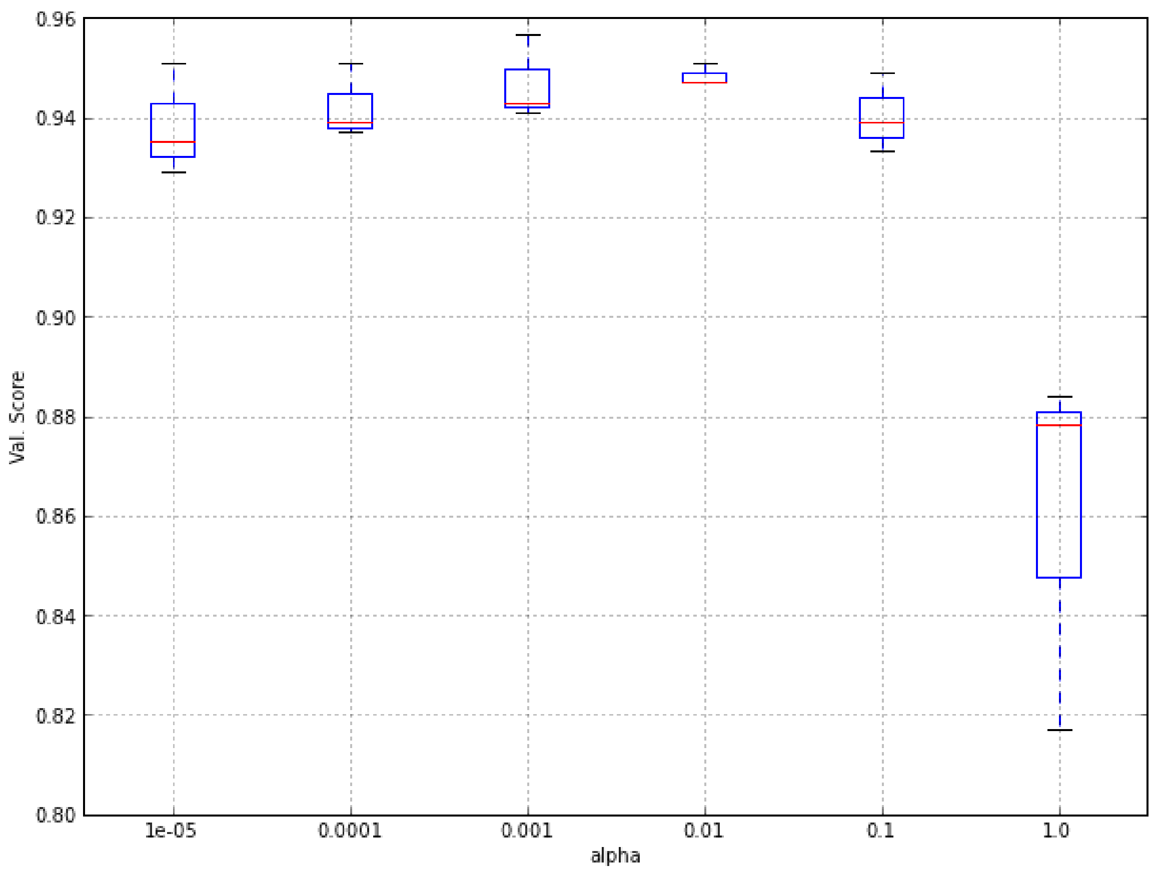
In [67]:
```
nb_params = {'alpha': np.logspace(-5, 0, 6)}
nb_search.launch_for_arrays(MultinomialNB(), nb_params,
    X_train_small_stripped, y_train_small_stripped, n_cv_iter=3)
```

Out[67]:
```
Progress: 11% (002/018)

Rank 1: validation: 0.88114 (+/-0.00295) train: 0.93574 (+/-0.00721):
 {'alpha': 1.0}
```

In [68]:
```
nb_search
```

Out[68]:
```
Progress: 100% (018/018)

Rank 1: validation: 0.94826 (+/-0.00131) train: 0.99716 (+/-0.00058):
 {'alpha': 0.01}
Rank 2: validation: 0.94695 (+/-0.00494) train: 0.99716 (+/-0.00058):
 {'alpha': 0.001}
Rank 3: validation: 0.94237 (+/-0.00429) train: 0.99781 (+/-0.00058):
 {'alpha': 0.0001}
Rank 4: validation: 0.94041 (+/-0.00458) train: 0.99191 (+/-0.00079):
 {'alpha': 0.10000000000000001}
Rank 5: validation: 0.93844 (+/-0.00645) train: 0.99781 (+/-0.00058):
 {'alpha': 1.0000000000000001e-05}
```

In [69]:
```
nb_search.boxplot_parameters(display_train=False)
```



For low values of alpha (no smoothing), the model is not biased and hence free to overfit. Smoothing a bit with `alpha=0.001` or `alpha=0.01` makes the validation score increase a bit (thus overfitting a bit less but not by much). If alpha is too strong the model is too biased or constrained and underfits.

Setting Up a Pipeline for Cross Validation and Model Selection of the Feature Extraction parameters

The feature extraction class has many options to customize its behavior:

In [70]:
```
print(TfidfVectorizer.__doc__)
```

```
In [70]: print(TfidfVectorizer.__doc__)
```

```
Convert a collection of raw documents to a matrix of TF-IDF features.

    Equivalent to CountVectorizer followed by TfidfTransformer.

    Parameters
    ----------
    input : string {'filename', 'file', 'content'}
        If filename, the sequence passed as an argument to fit is
        expected to be a list of filenames that need reading to fetch
        the raw content to analyze.

        If 'file', the sequence items must have 'read' method (file-like
        object) it is called to fetch the bytes in memory.

        Otherwise the input is expected to be the sequence strings or
        bytes items are expected to be analyzed directly.

    charset : string, 'utf-8' by default.
        If bytes or files are given to analyze, this charset is used to
        decode.
```

In order to evaluate the impact of the parameters of the feature extraction one can chain a configured feature extraction and linear classifier (as an alternative to the naive Bayes model:

```
In [71]: from sklearn.linear_model import PassiveAggressiveClassifier
         from sklearn.pipeline import Pipeline

         pipeline = Pipeline((
             ('vec', TfidfVectorizer(min_df=1, max_df=0.8, use_idf=True)),
             ('clf', PassiveAggressiveClassifier(C=1)),
         ))
```

Such a pipeline can then be cross validated or even grid searched:

```
In [78]: from sklearn.cross_validation import cross_val_score
         from scipy.stats import sem

         scores = cross_val_score(pipeline, twenty_train_small.data,
                                  twenty_train_small.target, cv=3, n_jobs=-1)
         scores.mean(), sem(scores)

Out[78]: (0.96116027531956727, 0.0026015253796112022)
```

For the grid search, the parameters names are prefixed with the name of the pipeline step using "__" as a separator:

```
In [80]: from sklearn.grid_search import GridSearchCV

         parameters = {
             #'vec__min_df': [1, 2],
             'vec__max_df': [0.8, 1.0],
             'vec__ngrams_range': [(1, 1), (1, 2)],
             'vec__use_idf': [True, False],
         }

         gs = GridSearchCV(pipeline, parameters, verbose=2, refit=False)
         _ = gs.fit(twenty_train_small.data, twenty_train_small.target)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-80-a4d7351dda45> in <module>()
      9
     10 gs = GridSearchCV(pipeline, parameters, verbose=2, refit=False)
---> 11 _ = gs.fit(twenty_train_small.data, twenty_train_small.target)

/Library/Frameworks/EPD64.framework/Versions/7.3/lib/python2.7/site-
packages/scikit_learn-0.13.1-py2.7-macosx-10.5-x86_64.egg/sklearn/grid_search.pyc
in fit(self, X, y, **params)
    370                                   self.loss_func, self.score_func,
self.verbose,
    371                                   **self.fit_params)
--> 372                               for clf_params in grid for train, test in cv)
    373
    374         # Out is a list of triplet: score, estimator, n_test_samples

/Library/Frameworks/EPD64.framework/Versions/7.3/lib/python2.7/site-
packages/scikit_learn-0.13.1-py2.7-macosx-10.5-
x86_64.egg/sklearn/externals/joblib/parallel.pyc in __call__(self, iterable)
    512         try:
    513             for function, args, kwargs in iterable:
--> 514                 self.dispatch(function, args, kwargs)
    515
    516             self.retrieve()

/Library/Frameworks/EPD64.framework/Versions/7.3/lib/python2.7/site-
packages/scikit_learn-0.13.1-py2.7-macosx-10.5-
x86_64.egg/sklearn/externals/joblib/parallel.pyc in dispatch(self, func, args,
kwargs)
    309         """
    310         if self._pool is None:
--> 311             job = ImmediateApply(func, args, kwargs)
    312             index = len(self._jobs)
    313             if not _verbosity_filter(index, self.verbose):

/Library/Frameworks/EPD64.framework/Versions/7.3/lib/python2.7/site-
packages/scikit_learn-0.13.1-py2.7-macosx-10.5-
x86_64.egg/sklearn/externals/joblib/parallel.pyc in __init__(self, func, args,
kwargs)
    133         # Don't delay the application, to avoid keeping the input
    134         # arguments in memory
--> 135         self.results = func(*args, **kwargs)
    136
    137     def get(self):

/Library/Frameworks/EPD64.framework/Versions/7.3/lib/python2.7/site-
packages/scikit_learn-0.13.1-py2.7-macosx-10.5-x86_64.egg/sklearn/grid_search.pyc
in fit_grid_point(X, y, base_clf, clf_params, train, test, loss_func, score_func,
verbose, **fit_params)
     82     # update parameters of the classifier after a copy of its base
structure
     83     clf = clone(base_clf)
---> 84     clf.set_params(**clf_params)
     85
     86     if hasattr(base_clf, 'kernel') and hasattr(base_clf.kernel,
'__call__'):

/Library/Frameworks/EPD64.framework/Versions/7.3/lib/python2.7/site-
```

```
packages/scikit_learn-0.13.1-py2.7-macosx-10.5-x86_64.egg/sklearn/base.pyc in
set_params(self, **params)
    235                                         (name, self))
    236                     sub_object = valid_params[name]
--> 237                     sub_object.set_params(**{sub_name: value})
    238                 else:
    239                     # simple objects case

/Library/Frameworks/EPD64.framework/Versions/7.3/lib/python2.7/site-
packages/scikit_learn-0.13.1-py2.7-macosx-10.5-x86_64.egg/sklearn/base.pyc in
set_params(self, **params)
    240                     if not key in valid_params:
    241                         raise ValueError('Invalid parameter %s ' 'for estimator
%s'
--> 242                                          % (key, self.__class__.__name__))
    243                     setattr(self, key, value)
    244             return self

ValueError: Invalid parameter ngrams_range for estimator TfidfVectorizer

[GridSearchCV] vec__max_df=0.8, vec__use_idf=True, vec__ngrams_range=(1, 1) ....
```

In [81]:
```
gs.best_score_
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-81-179e7fb439ec> in <module>()
----> 1 gs.best_score_

AttributeError: 'GridSearchCV' object has no attribute 'best_score_'
```

In [82]:
```
gs.best_params_
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-82-cdd9b47fcec7> in <module>()
----> 1 gs.best_params_

AttributeError: 'GridSearchCV' object has no attribute 'best_params_'
```

Introspecting Model Performance

Displaying the Most Discriminative Features

Let's fit a model on the small dataset and collect info on the fitted components:

In [83]:
```
_ = pipeline.fit(twenty_train_small.data, twenty_train_small.target)
```

In [84]:
```
vec_name, vec = pipeline.steps[0]
clf_name, clf = pipeline.steps[1]
```

```
feature_names = vec.get_feature_names()
target_names = twenty_train_small.target_names

feature_weights = clf.coef_

feature_weights.shape
```

Out[84]: (4, 34109)

By sorting the feature weights on the linear model and asking the vectorizer what their names is, one can get a clue on what the model did actually learn on the data:

In [85]:
```
def display_important_features(feature_names, target_names, weights, n_top=30):
    for i, target_name in enumerate(target_names):
        print("Class: " + target_name)
        print("")

        sorted_features_indices = weights[i].argsort()[::-1]

        most_important = sorted_features_indices[:n_top]
        print(", ".join("{0}: {1:.4f}".format(feature_names[j], weights[i, j])
                        for j in most_important))
        print("...")

        least_important = sorted_features_indices[-n_top:]
        print(", ".join("{0}: {1:.4f}".format(feature_names[j], weights[i, j])
                        for j in least_important))
        print("")

display_important_features(feature_names, target_names, feature_weights)
```

```
Class: alt.atheism

atheism: 2.8369, atheists: 2.7697, keith: 2.6781, cobb: 2.1986, islamic: 1.7952,
okcforum: 1.6646, caltech: 1.5838, rice: 1.5769, bobby: 1.5187, peace: 1.5151,
freedom: 1.4775, wingate: 1.4733, tammy: 1.4702, enviroleague: 1.4619, atheist:
1.4277, psilink: 1.3985, rushdie: 1.3846, tek: 1.3809, jaeger: 1.3783, osrhe:
1.3591, bible: 1.3543, wwc: 1.3375, mangoe: 1.3324, perry: 1.3082, religion:
1.2733, benedikt: 1.2581, liar: 1.2288, lunatic: 1.2110, free: 1.2060, charley:
1.2006
...
good: -0.8709, dm: -0.8764, 10: -0.8786, brian: -0.8900, objective: -0.8986, deal:
-0.9098, thanks: -0.9174, order: -0.9174, image: -0.9258, scic: -0.9314, force:
-0.9314, useful: -0.9377, com: -0.9414, weiss: -0.9428, interested: -0.9465, use:
-0.9525, buffalo: -0.9580, fbi: -0.9660, 2000: -0.9810, they: -1.0051, muhammad:
-1.0165, out: -1.0520, kevin: -1.0545, org: -1.0908, morality: -1.1773, mail:
-1.1945, graphics: -1.5805, christian: -1.6466, hudson: -1.6503, space: -1.8655

Class: comp.graphics

graphics: 4.3650, image: 2.5319, tiff: 1.9232, file: 1.8831, animation: 1.8733, 3d:
1.7270, card: 1.7127, files: 1.6637, 42: 1.6542, 3do: 1.6326, points: 1.6154, code:
1.5795, computer: 1.5767, video: 1.5549, color: 1.5069, polygon: 1.5057, windows:
1.4597, comp: 1.4421, package: 1.3865, format: 1.3183, pc: 1.2518, email: 1.2262,
cview: 1.2155, hi: 1.2004, 24: 1.1909, postscript: 1.1827, virtual: 1.1706, sphere:
1.1691, looking: 1.1613, images: 1.1561
...
astronomy: -0.9077, are: -0.9133, who: -0.9217, bill: -0.9354, atheism: -0.9397,
org: -0.9404, christian: -0.9489, funding: -0.9494, that: -0.9597, by: -0.9654,
```

```
solar: -0.9708, access: -0.9722, us: -0.9907, planets: -0.9992, cmu: -1.0507, moon:
-1.0730, you: -1.0802, nasa: -1.0859, dgi: -1.1009, jennise: -1.1009, writes:
-1.1152, was: -1.1369, beast: -1.1597, dc: -1.2858, he: -1.3806, orbit: -1.3853,
edu: -1.4121, re: -1.4396, god: -1.6422, space: -3.5582

Class: sci.space

space: 5.7627, orbit: 2.3450, dc: 2.0973, nasa: 2.0815, moon: 1.9315, launch:
1.8711, sci: 1.7931, alaska: 1.7344, solar: 1.6946, henry: 1.6384, pat: 1.5734,
ether: 1.5178, nick: 1.4982, planets: 1.4155, dietz: 1.3681, cmu: 1.3530, aurora:
1.3106, nicho: 1.2958, funding: 1.2768, lunar: 1.2757, astronomy: 1.2595, flight:
1.2418, rockets: 1.2048, jennise: 1.1963, dgi: 1.1963, shuttle: 1.1652, spacecraft:
1.1631, sky: 1.1593, digex: 1.1247, rochester: 1.1080
...
any: -0.8163, computer: -0.8183, gaspra: -0.8261, bible: -0.8342, video: -0.8485,
religion: -0.8640, format: -0.8682, fbi: -0.8720, com: -0.8725, card: -0.8737, cc:
-0.8828, code: -0.8875, 24: -0.8883, library: -0.8904, sgi: -0.9208, halat:
-0.9531, 3d: -0.9607, ___: -0.9630, points: -1.0150, tiff: -1.0278, color: -1.0560,
keith: -1.0664, koresh: -1.1302, file: -1.1529, files: -1.1679, image: -1.3169,
christian: -1.3767, animation: -1.4241, god: -1.7873, graphics: -2.5640

Class: talk.religion.misc

christian: 3.0979, hudson: 1.8959, who: 1.8842, beast: 1.8652, fbi: 1.6698, mr:
1.6386, buffalo: 1.6148, 2000: 1.5694, abortion: 1.5172, church: 1.5061, koresh:
1.4853, weiss: 1.4829, morality: 1.4750, brian: 1.4736, order: 1.4545, frank:
1.4508, biblical: 1.4123, 666: 1.3742, thyagi: 1.3520, terrorist: 1.3306,
christians: 1.3202, mormons: 1.2810, amdahl: 1.2641, blood: 1.2380, freenet:
1.2299, rosicrucian: 1.2122, mitre: 1.2032, christ: 1.1982, objective: 1.1635,
love: 1.1519
...
file: -0.9489, saturn: -0.9516, university: -0.9569, on: -0.9592, ac: -0.9685,
lunatic: -0.9820, for: -0.9882, orbit: -0.9893, some: -1.0031, anyone: -1.0355, uk:
-1.0703, liar: -1.0715, ibm: -1.0965, wwc: -1.1029, thanks: -1.1200, freedom:
-1.1455, nasa: -1.1951, free: -1.2008, thing: -1.2337, atheist: -1.2573, princeton:
-1.2966, cobb: -1.3150, keith: -1.4660, caltech: -1.4869, graphics: -1.5331, edu:
-1.5969, atheism: -1.7381, it: -1.7571, atheists: -1.9418, space: -2.2211
```

Displaying the per-class Classification Reports

In [86]:
```python
from sklearn.metrics import classification_report

predicted = pipeline.predict(twenty_test_small.data)
```

In [87]:
```python
print(classification_report(twenty_test_small.target, predicted,
                            target_names=twenty_test_small.target_names))
```

```
                    precision    recall  f1-score   support

       alt.atheism       0.87      0.78      0.83       319
     comp.graphics       0.93      0.96      0.95       389
         sci.space       0.95      0.95      0.95       394
talk.religion.misc       0.76      0.80      0.78       251

       avg / total       0.89      0.89      0.89      1353
```

Printing the Confusion Matrix

The confusion matrix summarize which class where by having a look at off-diagonal entries: here we can see that articles about atheism have been wrongly classified as being about religion 57 times for instance:

In [88]:
```
from sklearn.metrics import confusion_matrix

confusion_matrix(twenty_test_small.target, predicted)
```

Out[88]:
```
array([[250,   5,   7,  57],
       [  2, 375,   6,   6],
       [  2,  15, 375,   2],
       [ 32,   9,   8, 202]])
```

In [ ]: