1. To recap from our January 27 practicum session on elementary text classification, the following code will put you in a position to run a Naive Bayes text classifier on the movie_reviews corpus from nltk:

```python
>>> import nltk, random
>>> from nltk.corpus import movie_reviews
>>> docs = [(list(movie_reviews.words(fileid)), category)
...    for category in movie_reviews.categories()
...    for fileid in movie_reviews.fileids(category)]
>>> pos_docs = [d for d in docs if d[1]=='pos']
>>> neg_docs = [d for d in docs if d[1]=='neg']
>>> random.seed(0)  # seed the random number generator to get reproducible behavior across runs
>>> random.shuffle(pos_docs)
>>> random.shuffle(neg_docs)
>>> (train_set, test_set, dev_set) = (pos_docs[100:] + neg_docs[100:],
...    pos_docs[50:100] + neg_docs[50:100],
...    pos_docs[0:50] + neg_docs[0:50])

>>> my_words = ['good', 'great', 'excellent', 'bad', 'terrible']

>>> def small_document_features(document):
...    document_words = set(document)
...    features = {}
...    for word in my_words:
...        features['contains(%s)' % word] = (word in document_words)
...    return features
...
>>> featurizer = small_document_features
```
>>> train_featureset = [(featurizer(d), c) for (d,c) in train_set]
>>> dev_featureset = [(featurizer(d), c) for (d,c) in dev_set]
>>> test_featureset = [(featurizer(d), c) for (d,c) in test_set]
>>> classifier = nltk.NaiveBayesClassifier.train(train_featureset)

>>> print("Performance with small featureset:")
Performance with small featureset:
>>> print(nltk.classify.accuracy(classifier, test_featureset))
0.66

>>> classifier.show_most_informative_features(5)

Most Informative Features
contains(terrible) = True neg : pos = 3.3 : 1.0
contains(excellent) = True pos : neg = 3.0 : 1.0
contains(bad) = True neg : pos = 1.9 : 1.0
contains(bad) = False pos : neg = 1.5 : 1.0
contains(great) = True pos : neg = 1.4 : 1.0

So we can get above-60% test-set performance just with five carefully-chosen words! Here’s what you have to do:

(a) The above code sets the variable featurizer to be the function that featurizes the document into a set of presence/absence features. In the above code, this function is small_document_features(document). Write alternative code defining a different function that simply uses the 20 most frequent words seen in the movie reviews dataset, and report its performance. For this problem, I want you to convert all words into lowercase, and exclude words that are not then matched by the regex ^[a-z]+$.

Hint #1: the line
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
will create a word frequency distribution that can be used as a dictionary with word-type keys and integer-count values, and it works faster than explicitly iterating through all the movie words with a for loop. In this line, you can add an if clause right after the for clause that filters out words not matched by the regex.

Hint #2: The NLTK book seems to have a mistake in it: the line
all_words.keys()[:N]
will not give you the N most frequent words in the frequency distribution. Rather, you have to sort the words yourself. If you have dictionary d whose values are integers (e.g., a frequency count!), the following code will return a list of the dictionary’s key-value pairs in descending order of integer values from largest to smallest:
sorted(d.items(),key=lambda x: x[1], reverse=True)

For example:

```python
>>> d = {}
>>> d['a'] = 3
>>> d['b'] = 6
>>> d['c'] = 2
>>> sorted(d.items(), key=lambda x: x[1], reverse=True)
[('b', 6), ('a', 3), ('c', 2)]
```

(b) I actually split the entire movie reviews dataset into three parts: a *training set* and a *testing set*, but also a *development set*. The idea of a development set is that one sets aside a dataset that is not used to train the model but used to test out how well new ideas might work, avoiding the danger of OVERFITTING to one’s test set. Let’s check the performance of our five carefully chosen words on the development set:

```python
>>> import re
>>> counts = {}
>>> for (d,c) in train_set:
...     for word in d:
...         if not re.match('^[a-z]+$',word):
...             continue
...         if not word in counts.keys():
...             counts[word] = 0
...         counts[word] = counts[word] + 1
... >>> sorted_counts = sorted(counts.items(), key=lambda x: x[1], reverse=True)
```

```python
>>> my_words = [x[0] for x in sorted_counts[0:20]]
```

This accuracy is just about the same performance as we saw on the test set.

For this part of the problem, your job is to use the development set to try to improve on the 5-word Naive Bayes classifier. Just like in part (a), you get 20 `my_words = .../True` features to work with, but you can choose any 20 features you like. You can use the development set to test out any ideas you have in mind, but don’t use the test set until you have finalized your choice of 20 features. Show your work in trying different features with the development set, and once you’re satisfied with your 20 features, run the resulting classifier exactly once on the test set, and report its performance. **You will not be graded on how well your classifier performs**, but rather on correctly following instructions and trying interesting things out.

Turning in Problem 1: there are two parts to completing this assignment. First, download the Python program instantiating the code I give above, from [http://idiom.ucsd.edu/~rlevy/teaching/2015winter/lign165/homework_assignments/homework_](http://idiom.ucsd.edu/~rlevy/teaching/2015winter/lign165/homework_assignments/homework_).
I've indicated where you should insert your code for problems 1a and 1b. Once you've completed your code, rename it

```
alternative_document_features_<PID>.py
```

where `<PID>` is replaced by your PID. Copy the Python file to your home directory on the `ieng6.ucsd.edu` instructional compute servers, log on to the servers, and at the command line invoke the command

```
/home/linux/ieng6/ln165w/public/homework_turnin/turnin_hw3_1.sh \
alternative_document_features_<PID>.py
```

where, once again, `<PID>` is replaced by your PID. This will turn in your Python code.

Second, write up an account of your work for Part 2 of this problem, and send it to lign165-homework@ling.ucsd.edu as usual. Please include your PID on the written part of your homework assignment so that we can easily associated the two together!

2. The Naive Bayes implementation you’ve been using has been an instance of the **Bernoulli model** described in Chapter 13 of Manning, Raghavan, and Schütze 2008, where the probability of a document `d` given a class `c` is modeled as:

\[
P(d|c) = \prod_{f \in d} P(f|c)
\]  

(1)

where `f` is an **indicator** feature (e.g., `document contains 'excellent' = True`).

The alternative formulation of the Naive Bayes model is the **multinomial model**, in which the probability of a document `d` given a class `c` is modeled as:

\[
P(d|c) = \prod_{1 \leq k \leq n_d} P(t_k|c)
\]  

(2)

where \(\langle t_1, t_2, \ldots, t_{n_d}\rangle\) are the terms (words) in the document that are considered “in vocabulary” for classification purposes, and \(n_d\) is the number of such terms in the document. Note that the multinomial model is a **by-token model**: if a given term `t` (e.g., “excellent”) appears twice in a given document, it will contribute twice to the product in Equation \(\text{(2)}\). For example, for the mini-text

```
bit by bit
```

Equation \(\text{(2)}\) would be instantiated as

\[
P(d|c) = P(\text{bit}|c)P(\text{by}|c)P(\text{bit}|c)
\]

or equivalently

\[
P(d|c) = P(\text{bit}|c)^2P(\text{by}|c)
\]
Your job in this problem is to implement the multinomial Naive Bayes model. Download the Python code [http://idiom.ucsd.edu/~rlevy/teaching/2015winter/lign165/homework_assignments/homework_3/nb_multinom.py](http://idiom.ucsd.edu/~rlevy/teaching/2015winter/lign165/homework_assignments/homework_3/nb_multinom.py) and look at it. The first section of this file sets up a trimmed version of the movie reviews dataset: I’ve removed the stopwords (see Chapter 2 of the NLTK book) and then removed all but the first 50 words from each document. This is to simplify some of the numerical computation issues with Naive Bayes. The second section is a “toy” example that you might want to uncomment while testing your code: the toy example is small enough that you can work out by hand how Naive Bayes ought to work, and test your code against it. The third section is labeled `### Your solution should go here` function. You should put your solution in this third section of the code. You definitely need to change the internal contents of the function; you might also want to add some material outside the function definition to store conditional probabilities $P(w|c)$. The last line of the program tests the code on the first development-set document.

**Turning in Problem 2.** Once you’ve completed your implementation, rename the resulting Python file `nb_multinom_<PID>.py` where, as in Problem 1, `<PID>` is your PID. Copy the file to your home directory on the ieng6.ucsd.edu instructional compute servers, log on to the servers, and at the command line invoke the command

```
/home/linux/ieng6/ln165w/public/homework_turnin/turnin_hw3_2.sh 
 nb_multinom_<PID>.py
```

where, once again, `<PID>` is replaced by your PID. This will turn in your Python code. There’s no need for a separate writeup for this problem.