LING/CSCI 5832 Fall 2016 Homework 2 - language identification

Due Tue Sep 27 before class (2pm) on D2L.

In this homework, the task is to build a language identification system. That is, given some input text, it should identify which language that text is written in. Here, we will just distinguish between **English**, **German**, **Dutch**, and **Swedish** (which are all Germanic languages using similar symbols so the task is not trivial). In principle, the same system could be extended to hundreds of languages if training data were available. The system should use letter n-grams (specifically 2-grams). The idea is to implement a bigram language model that solves this task.

The Data

You are given four files where each contains just one line of lowercased text with spaces between words (no punctuation). That means you don't have to worry about tokenization or lowercasing and other low-level details. The four files represent training data in four languages (en = English, de = German, nl = Dutch, sv = Swedish). They look like this:

```
sv: är en inom familjen den är känd för att vara det djur ...
```

The data itself comes from a cleaned-up random Wikipedia dump.

Classification

The main thing to extract from the data are the conditional probabilities

$p(l_i|l_{i-1}, \text{class})$

that is, the probability of seeing letter l_i, given that the previous letter was l_{i-1}, in some language.

Having access to that information, we can build a language model that assigns probabilities to unknown sentences of length n letters by:

$p(\text{sentence}|\text{class}) \propto p(c) \times p(l_1|l_0,\text{class}) \times ... \times p(l_n|l_{n-1}, \text{class})$

Since we have four texts to learn from, one from each language, we can skip the class prior (it will be the same for all) and just calculate:

$p(l_1|l_0,class) \times ... \times p(l_n|l_{n-1}, class)$ for each class.

To make matters simple, you can assume all languages have the same alphabet (of size `32`, which is the number of unique letters in all texts). Now, $p(l_i|l_{i-1}, class)$ can be estimated by:

$[count(l_{i-1}\ l_i) + \delta] / [count(l_{i-1}) + \delta|V|]$

where **count($l_{i-1}\ l_1$)** is how many times you saw the sequence $l_{i-1}\ l_i$ in the training data, and **count(l_{i-1})** is how many times you saw the letter l_{i-1} in the training data.

Here, **|V|** is the size of the alphabet, and **δ** a smoothing "pseudocount" to avoid zeroes (setting it to 0.5 or 1.0 works OK). For example, the Dutch data never contains the sequence `zz`, but we wouldn't like to give **p(z|z, Dutch)** a zero probability, and using a pseudocount avoids this.

Use log probabilities

Because of potential underflow issues in multiplication, you should convert all probabilities to log probabilities. This of course means that multiplication becomes addition in logspace.

Use Python dictionaries to store probabilities per language

In actuality, you should create a dictionary in Python that contains the probabilities for each bigram and language. It could contain entries like this:

```
probs['de']['ba'] = -2.397835892138634
```

representing the log probability **p(a|b, German)**. Having such a dictionary, the task reduces to going over a string letter by letter and adding up the log probabilities. For example, if your string is "abracadabra", you would declare the first letter to be a space (for padding/the initial symbol), and add up:

```
probs['de'][' a'] + probs['de']['ab'] + probs['de']['br'] + probs['de']['ra'] + ...
```

to calculate the log probability of German having generated "abracadabra". If there is some two-letter sequence `xy` ~~that you never saw in the training data~~ where one, or both of the symbols `x` or `y` never appeared in the training data, i.e. that doesn't have a defined `probs[lang]['xy']` for any lang (even after smoothing), just skip it (assign log prob `0` - this is done conveniently and automatically by using Python dictionaries' get method, e.g. probs['de'].get(bigram, 0), where bigram is the two-letter sequence you're interested in.)

Code

After creating the dictionary, you should create a function called `classify` that takes as input a string (and possibly the dictionary "probs" and a list of classes), and returns a two-tuple of `(bestclass, probabilities)`, where `bestclass` is the highest probability language, and the probabilities is a dictionary containing the log probability assigned to each of the four classes.

The last four lines of your code could look something like this:

```
print classify(probs, classes, u'this is a very short text')
print classify(probs, classes, u'dies ist ein sehr kurzer text')
print classify(probs, classes, u'dit is een zeer korte tekst')
print classify(probs, classes, u'detta är en mycket kort text')
```

(If you're using Python 3, skip the `u` before the quoted text).

And your program overall should output should be something like the following, classfying each of the four above sentences in to one of the classes, and printing out the log probabilities.

```
('en', {'de': -80.35172540755437, 'en': -66.21428246521677, 'nl': -77.1037365908808
('de', {'de': -76.90839602299691, 'en': -94.36019776646441, 'nl': -84.4416945288086
('nl', {'de': -77.4986872052995, 'en': -84.71022310643905, 'nl': -67.6048558953818
('sv', {'de': -91.68806229644318, 'en': -104.62575320386907, 'nl': -95.47778310073
```

Don't worry about getting the exact same log probabilities as here (they might vary depending on the log base you're using and other details). But you should classify the four sentences correctly.

What to hand in

You should hand in an archive on D2L `firstname.lastname.5832.hw2.tar.gz` or `firstname.lastname.5832.hw2.zip` consisting at least of a Python file `firstname.lastname.5832.hw2.py` that runs as decribed if the language data files are located in the same directory and prints out something like the above. Put a `README` in the archive if you did anything outside a straightforward implementation of the above.

Addendum: note on coding technique

You should (obviously) not replicate code for the different languages. Instead you should store the language-dependent data in a dictionary, indexed by the language. For example, to read in all the files into a dictionary (in Python 2.7) you could do something like:

```
import codecs
classes = ['de','en','nl','sv']
texts = {}
alphabet = set()
for lang in classes:
    texts[lang] = [line.strip() for line in codecs.open(lang,'r',encoding='utf-8')]
    alphabet |= {unigram for unigram in texts[lang]}
```

Which would read in the files: `de, en, nl, sv`, and assign e.g. `texts['en']` to contain the entire English text, and build a set `alphabet` that contains all the symbols ever used in any of the languages. For Python 3.3, don't use the `codecs` package since that version of Python handles Unicode without special precautions.

After this, to create the dictionary that contains probs[lang][bigram]-type entries, you should also do this in a loop:

```
for lang in classes:
    # Look at text, collect counts, create probs[lang] dictionary...
```