Ling 473 Project 5
Due 11:45pm on Saturday, September 7, 2013

For this project you will build a naïve Bayesian classifier which is able to classify text fragments according to their language. Determining the language of a document may seem trivial, since the most common few words in each language—or the character set it uses—could be thought of as an identifying signature. But we would like to use a more principled approach that *quantitatively scores* the probability of some fragment of text being written in one language, relative to others.

Naturally, Bayes' theorem comes to mind:

$$P(\text{lang}|\text{text}) = \frac{P(\text{text}|\text{lang})P(\text{lang})}{P(\text{text})}$$

As we did in the POS-tagging lecture, we can ignore the denominator of the right side. This is because we will be interested in comparing the result of this calculation for different languages—but always for the same text. Since we will only compare results for the same text fragment, $P(\text{text})$ will always be the same value (whether we know what it is or not), and it can be excluded from the calculation.

$$P(\text{lang}|\text{text}) = P(\text{text}|\text{lang})P(\text{lang})$$

Next, we will assume that each of the 15 languages are equally likely. Again, we could always multiply the right side by $\frac{1}{15}$, but since we will only use the results for comparison, this term can also be dropped.

$$P(\text{lang}|\text{text}) = P(\text{text}|\text{lang})$$

Finally, we apply the naïve Bayesian assumption: all words in the text are *conditionally independent* of each other. Therefore, the probability that a text fragment is written in a language will be product of the probabilities of all of its words, according to a unigram language model of that language.

$$P(\text{lang}|\text{text}) = \prod_i P(\text{word}_i|\text{lang})$$

Because this product will be quite small and may cause underflow when using an approximation of real numbers like floating point, the calculation should be done by adding log-probabilities.

$$\text{logprob}(\text{lang}|\text{text}) = \sum_i \log P(\text{word}_i|\text{lang})$$

This calculation is repeated for each of the 15 languages, and the language with the highest probability is predicted. If we just wanted to know the best predicted language $\hat{L}$, we could use

argmax to drop the left side of the equation. As you recall, this notation discards the actual result, returning only the parameter that yielded it.

$$\hat{L} = \text{argmax}_j \sum_i \log P(\text{word}_i | \text{lang}_j)$$

However, for this project, in addition to identifying the best prediction, you will calculate and report the actual log-prob scores for each line of text, for each of the 15 languages.

Input

In the following location you will find the **corpus**: 15 unigram language model files similar to what you built for Project 2. The models are truncated, containing only the most common 1,500 words.

```
/dropbox/12-13/473/project5/language-models
```

These files use the Latin-1 encoding, an 8-bit file format. Because some languages use lexical capitalization (for example, nouns are always capitalized in German), and because some programming languages may lack support for correctly changing the case of letters with diacritics, this project will use case-sensitive comparison. Do not ignore case.

In order to match the way the language model was built, you should strip punctuation from the input sentences. A suggested set of punctuation to strip is given here: `.,!¡¥$£?¿;:()"'—–-/[]¹²³«»` but you are welcome to develop your own set, based on an examination of the language models.

In accordance with typical classification practice, you will be provided with separate training and testing input files. These files, **`test.txt`** and **`train.txt`** are located in the `/project5` dropbox. Each example is on a single line, consisting of an identifier, a tab character, and the example text. The **training** examples are labeled, so the identifier will be the correct three-letter language code for the example. The **test** examples are unlabeled, so the identifier will be a numeric value. Since the training examples are taken from the training corpus, they should all be readily identified by your classifier, and you can use them to verify that it is working properly. You only need to submit results for the testing examples.

Smoothing

As is usual, you will encounter a lot of words in the test data which are not in the language model that you are currently evaluating. How you deal with these is called *smoothing*, and you should implement a principled approach to the problem. One possibility is to assign unseen words the same probability as a singleton (a word that occurred once in the model) for the language under consideration.

Discuss your smoothing technique in your write-up and comment on the effect your choice has on your classifier's probability space.

Extra Credit

For extra credit (10 points), also process the file `extra-test.txt`, which includes samples from languages that are *not* included in your fifteen models. Using a threshold value or other parameters, you may determine that none of the languages is clearly more likely than any of the others. In this case, your classifier can conclude that it is an 'unknown' language. Use the `extra-train.txt` file, in which the languages are labeled, to tune your algorithm and/or parameter(s).

Output Format

Output your results to the console (stdout). For each example you process, write the following lines:

- The identifier, a tab character, and the example text
- One line per language model, consisting of the language code, a tab character and a numeric value
- The word "result", a tab character, and the language code for the "best" result from your classifier.

Example output follows. Your output file will contain one section like this for each of the input examples. The numeric values can represent whatever you wish, since they are only relative for the single example. Depending on how you set up your math, the "result" should be the language that has either the highest—or the lowest—value.

```
2       Cé agus, is féidir cur síos a dhéanamh mar sin ar an gcóras.
dan     -17.434282
deu     -17.735051
dut     -17.394591
eng     -16.219135
fin     -17.451900
fra     -17.711787
gla     -13.688687
ita     -17.291179
nob     -17.722804
pol     -17.778136
por     -16.278133
spa     -17.402331
swe     -17.856767
swh     -17.201888
tgl     -17.359011
result gla
```

Submission

Include the following files in your submission:

`compile.sh`	Contains command(s) that compile your program. If you are using python, shell scripts, or any other interpreted language that does not require compiling, then this file will be empty, or contain just the single line: `#!/bin/sh`
`run.sh`	The command(s) that run your program. Be sure to include compiled binaries in your submission so that this script will execute without first running compile.sh
`condor.cmd`	Condor control file, suitable for running your program as follows: `condor_submit condor.cmd`
`output.txt`	Captured console output (stdout) from running your program, this should be produced by running your Condor job.
`extra-credit.txt`	Extra credit output (optional)
`readme.{pdf, txt}`	Your write-up of the project. Describe your approach, any problems or special features, or anything else you'd like me to review. If you could not complete some or all of the project's goals, please explain what you were able to complete.
`(source code and binary files)`	All source code and binary files (jar, a.out, etc., if any) required to run and compile your program

Gather together all the required files, making sure that, for example, any PDF or other binary files are transferred from your local machine using a binary transmission format. Then, from the parent directory of the one containing your files, issue the following command to package them for submission.

```
tar -czf project5.tar <your project dirname>
```

Notice that this command packages all files in the current directory; do not include any top-level directories. Upload the file to CollectIt.

Grading

Correct results	30
Write-up	20
Code has tests	15
Runs as-is	15
Clarity, elegance, and readability of code	10
Followed submitting instructions	10

Citation

Philipp Koehn. *Europarl: A Parallel Corpus for Statistical Machine Translation*. MT Summit 2005. http://www.iccs.inf.ed.ac.uk/~pkoehn/publications/europarl-mtsummit05.pdf