Intro to IS Lab: Wikipedia Language Classification

For this assignment, we will be investigating the use of decision trees and boosted decision stumps to classify text as one of two languages. Specifically, your task is to collect data and train (in several different ways) some decision stumps/trees so that when given a 15 word segment of text from either the English or Dutch Wikipedia, your code will state to the best of its ability which language the text is in.

Data collection

The first step will be to collect some data. Note that just as the English Wiki has a "Random article" link, Dutch has a "Willekeurige pagina" operation which you will probably avail yourself of. Also, you should take the information about how your model will be tested into account when collecting your training set - that is, each example should itself be a 15-word sample, so you can get many samples from a single page, but you should also obtain a number of different pages, since they will represent different authors writing in the same language. If you'd like, you may share any raw data you collect with other students. **NOT FEATURES and NOT CODE! Raw text only!**

Then, you will have to decide on some features to use for learning. That is, you cannot feed in 15 words of text as an input, but rather features of the 15 words. Since we have decision trees, they should be boolean (or few-valued) features, but can be boolean questions about numeric features as well. So, some features you could use (these may or may not be valuable) could be "Does it contain the letter Q?" or "Is the average word length > 5?" You will need to come up with at least **five distinct features**. Note that you can't use the same numeric feature as ten different attributes and have that count, though you are certainly welcome to create several binary attributes out of one numeric feature. Of course, you will need to create the same features for your training data, your test data, and the novel test examples that I will provide when grading (see "What to hand in" below). If you don't know where to start consider looking for a few function words from each language.

Experimentation

You will implement two learning algorithms: a decision tree and Adaboost using decision trees as the learning algorithm.

You will need to write code that creates a decision tree for your data, **based on the information gain algorithm covered in the book and class**. You will also be implementing Adaboost using decision stumps. Remember that Adaboost relies on your learning algorithm to accept weighted examples. Keep this in mind when designing your decision tree algorithm.

In order to evaluate your algorithms, you should set aside a test set on which to test your algorithms performance. Using this you can determine the error rate of your algorithms given certain features, training sizes, and learning parameters (e.g. different depths and/or entropy cutoffs of the single decision tree, and different numbers of stumps when boosting). **This process should be fully explained in your writeup**.

For testing purposes your implementations should train on files with the format demonstrated here in train.dat. Each line is a training example consisting of a label (either "en" for English or "nl" for Dutch the format) followed by a '|' followed by 15 words. Notice that the training data could be in any order and begin in the middle of sentences and contain numbers and punctuation (other than '|'). Whether you use punctuation or just strip it all out is up to you. After learning a model you will then have to somehow save it to be loaded later by a classifier. The easiest way to do this is via serialization (Java or Python) but it is ok if you come up with some other scheme. The only requirement is that whatever you training program outputs your prediction program can load and use. Testing data will follow a similar format as above but with out the labels, as in test.dat. Each line is an observation without a label (nor the special delimiter '|')

that your program is expected to classify.

What to hand in

You should hand in the following:

- Some code! This can be written in Java, C++, Python, or another language if given prior approval. It should have at least two entry points (different modules or command-line args):
  - **`train <examples> <hypothesisOut> <learning-type>`** should read in labeled examples and perform some sort of training.
    - `examples` is a file containing labeled examples. For example.
    - `hypothesisOut` specifies the file name to write your model to.
    - `learning-type` specifies the type of learning algorithm you will run, it is either "dt" or "ada". You should use (well-documented) constants in the code to control additional learning parameters like max tree depth, number of stumps, etc.
  - **`predict <hypothesis> <file>`** Your program should classify each line as either English or Dutch using the specified model. Note that this must **not** do any training, but should take a model and make a prediction about the input. For each input example, your program should simply print its predicted label on a newline. For example. It should not print anything else.
    - `hypothesis` is a trained decision tree or ensemble created by your `train` program
    - `file` is a file containing lines of 15 word sentence fragments in either English or Dutch. For example.
- Your best tree or ensemble in some hard-coded form (a serialized object, xml, etc), to enable the prediction. This is the `hypothesis` we will test using your `predict` function
  Addendum (April 17): your best tree should be named, `best.model`.
- Your examples, so we can test your training process.
- Documentation about how to use your code
- A writeup, containing:
  - a description of your features and how you chose them
  - a description of the decision tree learning, how you came up with the best parameters (max depth, etc.) and your own testing results
  - a description of the boosting, how many trees turned out to be useful, and your own testing
  - anything else you think we might like to know

Grading

- Decision tree implementation: 25%
- Adaboost implementation: 25%
- Example collection, feature selection, and evaluation/testing: 15%
- Training processes: 10%
- Correct predictions: 10%
- Writeup: 15%

For the correct predictions, I will present ten examples of each language. Generally your grade will be proportional to the number of correct predictions, however if your model just always predicts the same language will get less than half the credit.