COSI 134a (Fall 2014) Programming Assignment #1: Naïve Bayes Classifier
Due: October 1 2014, 23:59 EDT

Assignment

You are to write a Naïve Bayes classifier and apply it to a text classification task (predicting the genders of the authors of blog posts). You must implement training and classification methods for your classifier, design and implement features specific to the given corpus, and report on your results. You are to submit a zip or tarball to Latte containing your code, a 1-2 page report (as a PDF file), and a model file for your best-performing classifier (see below).

Background

A supervised text classifier such as Naïve Bayes uses a trained statistical model to infer or predict a label (also called a 'class' or 'category') for an unlabeled document (or 'instance'), given a vector of features. You should be clear on all of these concepts going into this assignment. If you are not, the class handout for lecture 3, §7.2.1 and the introduction to chapter 16 of Manning & Schütze, §6.5 of the NLTK book, and even the Wikipedia article on *Naive Bayes classifier* may be helpful.

Corpora

In this assignment, the documents you are responsible for classifying are blog posts; the labels will be 'M' for 'male' or 'F' for 'female', designating the gender of the post's author. There are 3,232 blog posts in the supplied corpus, which should be enough data to achieve reasonable results (see below for specifics) without taking unduly long to train.

In addition to the blog gender corpus, a corpus of personal names is also provided, each of which is labeled as 'male' or 'female'. There are 5,001 female names and 2,943 male names, making this corpus 'larger' than the blog corpus; but each document consists of just one word, which makes dealing with them substantially faster and easier. You may find it helpful to develop your classifier first with the names corpus and then move on to the blog post, but accurately classifying names is not a requirement for this assignment.

Distribution

The tarball distributed with this assignment includes the raw corpus data and a few Python modules to help you get started. The included code is reasonably well documented (please read it!) and has been tested, but if you find anything unclear or incorrect, please let us know right away. If you want to just ignore it and start from scratch, that's fine, but not recommended.

The `classifier` module contains an abstract base `Classifier` class that you should use as a superclass for your Naïve Bayes classifier. An example subclass called `TrivialClassifier` may be found in the `test_classifier` module; it doesn't do much, but it illustrates the basic interface. You should not need to change anything in either of those two modules.

Your classifier should go in the `naive_bayes` module, which is just a skeleton in the distributed code. Your job is to flesh out that skeleton. The included `test_naive_bayes` module includes some basic tests, including ones for baseline classifier performance; they will obviously fail at first, but should all pass by the time you're done.

The `corpus` module includes a base `Document` class and several types of corpus readers. Of these, the `NamesCorpus` and `BlogsCorpus` classes are the only ones that you should need to use directly. You will be responsible for writing subclasses of `Document` that include `features` methods tailored to the blog gender corpus, like the examples in the `test_naive_bayes` module.

You may (but need not) add your own modules to the ones provided. You may also (but need not) use any third-party libraries you wish (e.g., NLTK, NumPy), but you cannot just import (or copy!) a third-party Naïve Bayes classifier; its implementation must be entirely your own.

Evaluating Your Classifier

Just implementing a classifier is not insufficient—you have to show that it works, and then make it work better! The tests in the distributed `test_naive_bayes` module check the accuracy (i.e., the proportion of correctly classified documents) for the two gender corpora, as well as for a 'pseudo-corpus' of integers trivially classified as 'even' or 'odd'. The values used in those tests—100% for the integers, 70% for the names corpus with the supplied features, and 55% for the blogs corpus with bag-of-words as features—are baselines that your Naïve Bayes classifier should be able to achieve.

Once your classifier is working (i.e., it can pass those baseline tests), your next task is to improve its performance on the blog gender corpus. A state-of-the-art classifier (not Naïve Bayes) with linguistically sophisticated features can achieve close to 90% accuracy (see the paper *Improving Gender Classification of Blog Authors* by Mukherjee and Liu). For this assignment, you should aim for about 70%, but don't worry if you can't quite achieve that goal.

Experiment with the performance effects of varying training/test size splits and with using different smoothing techniques. Include your observations on these experiments in your report.

If you would like to use more detailed classifier evaluation metrics than accuracy (e.g., precision, recall, F-measure) when reporting your classifier's performance, feel free to do so. It's likely to be required for future classification assignments, but is not for this one; if you want to keep it simple, you can stick with just accuracy.

Feature Engineering

You'll quickly find that you need better features than just bag-of-words to do significantly better than baseline for the blog gender classification task. The paper by Mukherjee and Liu should give you some ideas, but try some simple things first: experiment with tokenization, n-grams, frequency counts, etc.

Choosing good features can be a tricky business. It can also be frustrating: sometimes you'll have what you think is a great idea for a feature, spend a while implementing it, and then discover that it lowers the performance of a classifier rather than raising it, or just takes too long to compute. Be patient, and don't get discouraged. Describe the kinds of features you tried in your report. You need not (but may) include all of the features you experiment with in your final code; it's okay to throw away ideas that don't pan out.

Once you have a set of features you're happy with, write up your classifier's performance, and then save a model trained using those features to a file using the `Classifier.save` method (or your replacement for it). Include that file along with your code in your submission. Make sure that your final evaluation metrics are reproducible by loading your saved model into the final version of your classifier—we'll be checking!

Grading

Your grade will be based on the correctness of your classifier's training and classification algorithms (60%), its performance on the blog gender corpus with your best set of features (10%), code clarity and style (10%), and the quality of your report (20%).