Scoring Procedure and Documentation of PPM Implementation

November 4, 2011

1 Working With the PPM Implementation

Working with `ppm.py` is pretty straightforward. The class that abstracts the PPM is called `Compressor`, and there are really only two relevant methods: `add()` and `score()`. These both do exactly what you would expect. Here's an example program:

```
import ppm
from __future__ import with_statement

if __name__ == '__main__':
    c = ppm.Compressor()

    # Train on every line in file
    with open('training_data') as f:
        for l in f.readlines():
            c.add(l)

    # Test on every line in another file
    with open('dev_data') as f:
        for l in f.readlines():
            c.score(l)
```

You can train it on pretty much any structured data, though of course your scoring results will depend on things like entropy and the features you're handing it.

2 Proposed Scoring Procedure

The procedure for scoring, at this point is as follows. To begin, we partition our data into two subsets: emails which occur over *at least* two red vertices (since emails can be addressed to lots of people), and those which are not. If we denote all of our data with D, then the red-red partition can be called $\mathcal{RR} \subset D$ and the other (nonred-nonred) data can be called $\mathcal{NN}$.

We train *two models*—one using only the $\mathcal{RR}$ partition, and the other using the $\mathcal{NN}$ partition. We can do so using the techniques delineated in the code above.

After scoring, we have two models that output log probs. The goal now is to take these two log probs and get a "score" for the content. The first part of this process is to map it

to the space of numbers $\in [0, 1]$ (*e.g.*, turn it into a probability). Trivially, we can do so by raising it over some constant, usually something like e or 2: $\text{PROB} = 2^p$.

Each model outputs a probability, let's call them $p_{\mathcal{RR}}$ and $p_{\mathcal{NN}}$. The next thing to do is to normalize these two probabilities so that they add to one, and then emit the probaility that a given bit of content is $\in \mathcal{RR}$. Also trivial, we can express this as $p_{\mathcal{RR}}(\text{content}) = \frac{p_{\mathcal{RR}}}{p_{\mathcal{RR}} + p_{\mathcal{NN}}}$. This gives us a balanced notion not just that a piece of content looks like it's $\in \mathcal{RR}$, but also that it looks like it's *not* $\in \mathcal{NN}$.

The complete scoring mechanism looks like so:

$$\frac{2^{\text{logprob}_{\mathcal{RR}}}}{2^{\text{logprob}_{\mathcal{RR}}} + 2^{\text{logprob}_{\mathcal{NN}}}} \tag{1}$$