Computational Linguistics I *CMSC723, Fall 2010*

P3: Machine Learning

Hand in at: `http://www.cs.utah.edu/~hal/handin.pl?course=cmsc723`.

Introduction

This project is about getting used to machine learning techniques as applied to NLP problems. We will not actually be implementing any ML techniques: we'll only use off-the-shelf toolkits. The goal is for you to get used to these toolkits, as well as to how regularization and data set size affect each other. Plus, of course, the elephant in the room: feature engineering.

1 Tool Warm-Up *(0%)*

We will be using three tools for this project, which you'll have to either install yourselves, or use the versions that I've put on both UMIACS and junkfood filesystems. They are:

- megam: a tool for linear classification, based either on maximum entropy or perceptron optimization: `http://www.umiacs.umd.edu/~hal/megam/`. Available as `~hal/bin/megam` on either UMIACS or Junkfood machines.
- FastDT: a tool for decision tree classification, based on information gain: `http://www.umiacs.umd.edu/~hal/FastDT/`. Available as `~hal/bin/FastDT` on UMIACS or Junkfood machines.

You can look at the web pages for the tools to find out the expected input formats. FastDT and megam are particularly good for NLP problems where we often have high-dimensional sparse, binary feature vectors. Basically you just put the class label, and then list all the active (i.e., non-zero) features as strings.

Note: megam **can** *take feature values that are not one. You need to specify* `-fvals` *to megam to do this. FastDT can* **only** *handle binary features: sorry!*

I've provided the sentiment (movie review) data in megam/FastDT format in `sentiment.tr` (for training) and `sentiment.de` (for development). I'm not giving you test data because this is just to get you warmed up.

To make sure all the tools are running, you can try some of the following commands:

```
% FastDT -maxd 5 sentiment.tr > sentiment.dt
Loading data from sentiment.tr...
1600 examples, 25616 features
Building model....515...

% head sentiment.dt
1
1
N bad
N worst
N 25
L 3 0
N well
```

```
N got
L 0 24
L 8 26

% FastDT -load sentiment.dt sentiment.de > /dev/null
Loading model from sentiment.dt...
Predicting on sentiment.de...
Error = 139 / 400 = 0.3475
```

In the first line we're training (maximum depth of 5), the next shows the top of the tree (best feature == "bad") and the last line does prediction. It'll print out the predictions for each example unless you redirect somewhere.

```
% megam binary sentiment.tr > sentiment.w
Scanning file...1600 train, 0 dev, 0 test, reading...done
it 1   dw 5.192e-06 pp -4.18188e-03 er 0.50062
it 2   dw 6.365e-03 pp -4.24746e-03 er 0.33813
it 3   dw 1.707e-03 pp -4.26663e-03 er 0.32937
it 4   dw 8.855e-03 pp -4.29850e-03 er 0.28750
it 5   dw 2.555e-03 pp -4.31422e-03 er 0.27813
...
it 99  dw 6.881e-02 pp -4.60505e-03 er 0.00375
it 100 dw 5.129e-04 pp -4.60510e-03 er 0.00250

% megam -predict sentiment.w binary sentiment.de > /dev/null
done
Error rate = 77 / 400 = 0.1925
```

We can see that with this, the logistic regression (megam's default) does a lot better than FastDT.

If we want to look at the weights that megam produces, it's helpful to sort them by their weights. We can do this using unix sort as:

```
% sort -k2,2gr sentiment.w | head -n5
quite 1.45502734184265136719
you 1.44054222106933593750
great 1.42284429073333740234
very 1.31058967113494873047
well 1.26265859603881835938

% sort -k2,2gr sentiment.w | tail -n5
plot -1.49658524990081787109
nothing -1.55031025409698486328
worst -1.56467199325561523438
any -1.69367289543151855469
bad -3.27668237686157226562
```

Interestingly, both megam and FastDT thought that the "best" feature was the word "bad", though of course this isn't guaranteed to happen.

2 Regularization and Quantity of Data *(40%)*

Decision trees have an obvious regularization parameter: the depth. This is settable by `-maxd #` in FastDT. For megam, the regularization parameter is λ, which controls the weight of the regularizer. You can set it by `-lambda #`, which defaults to 1 if it's not specified.

Note: for all of these questions you are asked what you expect to observe and what you actually observe. I highly doubt that they will always be the same, even if your expectations are spot on. So don't assume that you're doing something wrong just because things don't work out how they "should."

(WU1, 10%**)** Train decision trees of different depths. Try $\{1, 2, 3, 4, 6, 8, 10, 15, 20, 25, 30\}$. Produce a plot with maximum depth on the x-axis, and error rate on the y-axis. Show both training error rate as well as development error rate. What do you *expect* to observe? What do you *actually* observe? If it's different than what you expect, try to justify why.

(WU2, 10%**)** Do the same for megam with different regularization parameters. Try λ values of $\{0, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100\}$. Produce a similar plot as above, and again say what you expect to observe and what you actually observe, and how they relate.

We might expect some interplay between amount of data and regularization. First we'll produce learning curves. To do this, we will select subsets of the training data and use only that subset to train on. When I say to use 100 examples, please use the *FIRST* 100 examples.

(WU3, 10%**)** Produce a learning curve (x-axis is the number of examples, y-axis is development error rate) for both DT and megam. For both, use the default regularization. Try with the following numbers of examples: $\{5, 10, 20, 50, 100, 200, 400, 800, 1600\}$. What do you expect to observe and what do you actually observe?

Finally, we'll look at the interplay between regularization and data set size.

(WU4, 10%**)** For each of the training set sizes above, find the regularization parameter (from the set of possible regularization parameters above) for each algorithm that makes it do *best* on the *development data*. (If you have to break ties, break it in favor of the *more regularized* version.) Produce a plot where the x-axis is data set size and the y-axis is optimal regularization parameter, for each of the learning algorithms. Again, what do you expect to observe and what do you actually observe?

3 Feature Engineering *(60%)*

For the feature engineering part of this assignment, you will be working with the gender classification task. You will have 10000 training examples (half written by men, half by women), where each example is a blog post. You will be given 2000 test examples, which are roughly evenly balanced (but not necessarily exactly balanced). Your job is to predict as well as possible on these test examples.

In addition to the blog posts, I will also give you parse trees, as produced by the Berkeley parser. Feel free to use these in any way you want.

All of this data is available at `http://www.umiacs.umd.edu/~hal/courses/2010F_CL1/out/gender.tar.gz` so that this file didn't get too big. See the `README` file in that data for a brief explanation of the data.

As you might expect, I only care how well you do on the test data. (Hint: you'll probably want to separate out some of the training data as development data.) You can evaluate, ala all the other assignments, your guesses at `http://www.cs.utah.edu/~hal/tmp/gender.pl`.

As a baseline, I've produced a bag-of-words feature extractor in `defaultExtractor.pl`. If you've downloaded the gender data into the current directory, you should be able to run:

```
% defaultExtractor.pl > gender.all
% wc -l gender.all
10000 gender.all

% head -n9000 gender.all > gender.tr
% tail -n1000 gender.all > gender.de
% megam binary gender.tr > gender.w
Scanning file...9000 train, 0 dev, 0 test, reading...done
it 1   dw 7.326e-02 pp -9.38119e-04 er 0.44778
it 2   dw 5.252e-01 pp -9.45764e-04 er 0.35789
it 3   dw 3.176e+00 pp -9.53332e-04 er 0.29944
it 4   dw 6.218e-03 pp -9.53719e-04 er 0.29633
it 5   dw 5.009e+00 pp -9.62502e-04 er 0.26756
...
it 99  dw 6.549e-01 pp -1.00042e-03 er 0.00322
it 100 dw 1.916e-01 pp -1.00035e-03 er 0.00333

% sort -k2,2gr gender.w | head -n5
realizing 1.25800311565399169922
mountains 1.14106214046478271484
draft 1.11654388904571533203
seattle 1.10156202316284179688
private 1.09603226184844970703

% sort -k2,2gr gender.w | tail -n5
soo -1.27449774742126464844
disorder -1.30078911781311035156
heather -1.34676074981689453125
yoga -1.38794815540313720703
anna -1.39123630523681640625
```

In this data, class 1 is male and class 0 is female. What this means is that men talk about mountains and seattle, and women talk about yoga and disorders. Or something like that.

You might thing this looks really good: we're getting less than one percent error! But if you run on dev data:

```
% megam -predict gender.w binary gender.de > /dev/null
done
Error rate = 339 / 1000 = 0.339
```

You'll see that life isn't so good. We've *way* overfit the training data.

(For what it's worth, this gets roughly equivalent performance on the test data: 36.6%.)

Here's a big hint: I'm giving you the parse trees, and parts of speech *for a reason.* One interesting fact is that men and women tend to use different parts of speech at different rates. They also use different grammatical constructions, POS sequences, etc., at different rates.

(WU5 20%**)** Describe what you did.

When you upload your test data or hand it in, it should be one example per line (so a 2000 line file) and the first white-space separated term should be the predicted class. You can print whatever you want, but it had better be a number, and anything < 0.5 will be considered 0 and anything ≥ 0.5 will be considered 1. Futhermore, anything that's NOT a number (i.e., doesn't match the regexp `/[0-9][0-9\.]*/`) will be considered to be 0.

Hint: You may find it useful to use the CFG rule extractor from the previous project if you don't want to process the trees yourself.

Grading: Just like before, on a tiered system. Top three teams will get 10%, 8% and 5% extra credit, respectively. This is all based on your test data performance.

- $35\% \leq$ error rate $< 37\%$: 10%
- $33\% \leq$ error rate $< 35\%$: 20%
- $31\% \leq$ error rate $< 33\%$: 28%
- $29\% \leq$ error rate $< 31\%$: 32%
- $27\% \leq$ error rate $< 29\%$: 35%
- $25\% \leq$ error rate $< 27\%$: 37%
- $24\% \leq$ error rate $< 25\%$: 38%
- $23\% \leq$ error rate $< 24\%$: 39%
- error rate $< 23\%$: 40%

4 What To Hand In

Please hand in the following:

- `writeup.pdf` – A document answering all the WU questions.
- `partners.txt` – A file listing names and IDs of all members of your team.
- `gender.pred` – Your predictions on the test data (should be 2000 lines long)