Report for HW4

Linan Qiu

`lq2137`

Quick Start

To run everything, place everything from the `hw3.tar.gz` tarball (without enclosing folder) into a directory. Also place all `.java` files in. The directory should contain

- `run.sh`
- `TaggerTest.java`
- `Eval.java`
- `PyTaggerDecoder.java`
- `PyTaggerHistoryGenerator.java`
- `Tagger.java`
- `TaggerQ4.java`
- `TaggerQ5.java`
- `tagger_history_generator.py`
- `tagger_config.pyc`
- `tagger_decoder.py`
- `tag_dev.dat`
- `tag.model`
- `pipe_servers.py`
- `example.sent`
- `eval_tagger.py`
- `tag_dev.key`
- `tagger_config.py`
- `tag_train.dat`
- `make_dict.py`
- `history.scores`

Then run

```
$ sh run.sh
```

The `.sh` file will contain the rest of the instructions. In particular, the performance results for Q4 Q5 and Q6 will be displayed. In addition, as per requirements,

- Output Dev Data
  - `tag_dev_q4.out` is the tagging for q4
  - `tag_dev_q5.out` is the tagging for q5
  - `tag_dev_q6.out` is the tagging for q6
- V Models
  - `suffix_tagger.model` is the model for q5
  - `additional_tagger.model` is the model for q6

Question 4

Using the given v model, I got the following results. They are quite good.

```
2226 2459 0.905246034974
```

Question 5

We use an additional suffix feature that takes into account the last 3 characters of the word.

```
2303 2459 0.936559577064
```

Score improved significantly. This is natural, since we are taking into account additional features and the suffix of words do relate to the tags (eg. er, ing, etc)

Question 6

Features For Single Words

This is inspired by Collins (Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms).

Using the following features

- Previous Word `w_{i-1}`
- Word Two Back `w_{i-2`
- Next Word `w_{i+1}`
- Next Two Ahead `w_{i+2}`

This yielded the following results.

```
2322 2459 0.944286295242
```

Features for Word Bigrams

Using the following features

- Previous Word `w_{i-1}`
- Word Two Back `w_{i-2`
- Next Word `w_{i+1}`
- Next Two Ahead `w_{i+2}`
- Word Bigram 1 `w_{i-2}, w_{i-1}`
- Word Bigram 2 `w_{i-1}, w_i`
- Word Bigram 3 `w_i, w_{i+1}`
- Word Bigram 4 `w_{i+1}, w_{i+2}`

Which is the word unigram features from the previous part plus word bigram features, we obtained the following result

```
2335 2459 0.949572997153
```

Features for Prefixes

We further add prefixes of length 4 or less. We tried using length 3 or less, but that somehow reduces the score. So we used length 4 as inspired by a certain paper that I forgot the author's name. P-something. Too late to look up again. Deadline's coming.

Using the following additional features

- Current Word `w_i`
- Previous Word `w_{i-1}`

- Word Two Back `w_{i-2`
- Next Word `w_{i+1}`
- Next Two Ahead `w_{i+2}`
- Word Bigram 1 `w_{i-2}, w_{i-1}`
- Word Bigram 2 `w_{i-1}, w_i`
- Word Bigram 3 `w_i, w_{i+1}`
- Word Bigram 4 `w_{i+1}, w_{i+2}`
- Prefixes of length 4 or less (originally used 3. It reduced score. Using 4 increased score)

```
2338 2459 0.950793005287
```

Word Form

The form of a word should affect tagging as well (eg. if the word is only numeric, or 4 numbers indicating possibly year). Hence, we use Bickel's rare word buckets as an indicator of the "form" of a word.

Using word buckets

```
2345 2459 0.953639690931
```

This improved results yet again! Bravo!