11-4/611 Assignment 6: POS Tagging

Assigned: 28 Feb 2020
Due: 26 Mar 2020 11:59PM

1 Background

POS tagging forms an important part of NLP workflow for most modern day NLP systems. Within the NLP community, POS tagging is largely perceived as a solved problem, or at least well enough solved such that most people don't put much effort into improving POS tagging for its own sake.

But clearly you didn't solve it, so you can take a crack at it.It's a good problem for learning about NLP, and that's what we are aiming for in this exercise.

2 Files

Download the handout from Autolab, and unzip the archive. You should have the following files:

```
handout
 ├── train_hmm.py
 ├── viterbi.py
 ├── tag_acc.py
 ├── ptb.2-21.tgs
 ├── ptb.2-21.txt
 ├── ptb.22.tgs
 ├── ptb.22.txt
 └── ptb.23.txt
```

For this assignment, we've provided you the text of the first 23 sections of the Penn Tree Bank (PTB). We've also provided you with the gold standard tags of the first 22 sections as well (one seems to be missing hmm). The first 21 sections are concatenated into `ptb.2-21.txt` and `ptb.2-21.tgs` for convenience. You will use section 22 and 23 for evaluation purposes in later tasks.

`ptb` datasets (files) refer to the Penn Tree Bank.

3 Task 1: POS tagging with HMM and Viterbi algorithm (100 points)

In this task you will be using an HMM to perform POS tagging. You are constrained to use an HMM, and you can only train on the provided training data (no external resources/libraries allowed). You should train your model with `ptb.2-21.txt` and `ptb.2-21.tgs`, test the model with `ptb.22.txt` and evaluate your model with `ptb.22.tgs`.

3.1 Subtask 1: training your HMM (30 points)

In this subtask, you will be training your bigram HMM (which is different from training a pokemon). Remember, a bigram HMM is one where the transition probability is defined as $\mathrm{P}(q_i|q_j)$, for $q_i, q_j \in Q$, the set of states. You should write your implementation in `train_hmm.py`. Your implementation should:

1. Read from command line the input file for text, the input file for tags, and an output file to write to.
2. For every pair of states $q_i, q_j \in Q$, calculate transition probability $\gamma q_i, q_j$. You can ignore smoothing for this task.
3. For every state $q_i \in Q$ and every token x_t in the text file, calculate emission probability $\eta q_i x_t$. You can ignore smoothing for this task.
4. Write the output to an output file

Take a look at `train_hmm.py`, some of the code has been implemented. You should fill in the sections commented with TODO. Further instructions are in the `train_hmm.py` file itself.

To run `train_hmm.py` from the command line,

```
$ python train_hmm.py tagfile.tgs textfile.txt my.hmm
```

tagfile.tgs is the tag file for training. textfile.txt is the token file for training. my.hmm is the output file that the `train_hmm.py` will write to.

3.2 Subtask 2: implementing Viterbi Algorithm(70 points)

In this subtask, you will be implementing Viterbi algorithm (which everyone learned from lecture and remembered well). Remember that Viterbi algorithm takes input a sequence of string tokens, and outputs a sequence of pos tags. Your implementation should:

1. Read from command line the HMM model, the input file for text, and an output file to write to.
2. Use the Viterbi algorithm to calculate the best pos tag sequence corresponding to each line of the text.
3. Write the output to an output file

Take a look at `viterbi.py`, some of the code has been implemented. You should fill in the sections commented with TODO. Further instructions are in the `viterbi.py` file itself. To run `viterbi.py` from the command line,

```
$ python viterbi.py model.hmm input.txt myoutput.tgs
```

where model.hmm is the hmm file generated (hopefully by your `train_hmm.py`), input.txt is the file of text tokens, and myoutput.txt is the output file to write your pos tags into.

3.3 Evaluating your model

Use the provided `tag_acc.py` to obtain an accuracy score.

```
$ python tag_acc.py gold-tags.tgs your-tags.tgs
```

Where the gold-tags.tgs is the file of correct tags we provide, and your-tags.tgs is the tags file generated by your `viterbi.py`. Do not modify the `tag_acc.py` file.

3.4 Submission

Include your modified `train_hmm.py` and `viterbi.py` in your submissions. Your program should follow the command line formats in section 3.1 and 3.2.

```
$ python train_hmm.py tagfile.tgs textfile.txt my.hmm
```

```
$ python viterbi.py model.hmm input.txt myoutput.tgs
```

4 Task 2: Building a Better System (0 points)

Note: this part is not a graded portion of the homework. It is only for those who are curious about how to improve an existing HMM model.
Woohoo! You have implemented a bigram HMM model! Clearly we can do better!^{TM} Now it is time to improve the model. You are still constrained to using a HMM and training on the provided training data (no external resources/libraries allowed). Some relatively simple but cost-effective measures include implementing a trigram HMM, or smoothing the probability estimates.

Feel free to make use of or modify the existing scripts provided for your needs.

Measure the accuracy of your new model against the same dataset as in the previous task.

Once you feel satisfied with the improvements, run and test your shiny new tagger on `ptb.22.txt`.

4.1 Questions

1. What modifications did you make?
2. How much improvement did your new model deliver?
3. Why do you think that these modifications improved the accuracy of this labelling task?

4.2 Reminders

Please check that your improved model is actually an improvement.

On the other hand, a positive improvement is an improvement no matter how small ☺

4.3 Frequently Encountered Problem

Your improvements may reduce the throughput of your tagging program by an order of magnitude. While we do not grade you on your code efficiency or speed, in the interest of your personal productivity, consider sampling a smaller but appropriate amount of data to test your system.