Ling/CSC 439/539: Assignment #3 (75 pts) (Graduate students)

By

Mithun Paul

General Software Requirements:

The following packages need to be installed for my code to run.

1. Pytorch
2. pandas
3. numpy
4. Python 3.6.3
5. scikit-learn
6. Nltk
7. Scipy
8. tqdm
9. If you are using **Conda** you can use the following commands:
    a. conda create **--**name mithunhw3 python=3 numpy scipy nltk scikit-learn pandas tqdm
    b. source activate mithunhw3
    c. conda install pytorch torchvision -c soumith
    d. pip install torchwordemb

Qn 1)

Ans:

Assumptions:

1. Assumes that there is a data folder in the current working directory and its contents are the 3 files with the same names as given in D2L

Steps to run:

1. after source activate mithunhw3
2. cd to directory q1
3. python initializer.py
4. Follow instructions on screen

Data:

1. Creates and trains a model on the training data
2. Saves a pickle of the training data. These are the various transition counts and probabilities. These can be found to be created in the same directory as the current working directory.
3. If you want to use a premade pickle, the pickles named *_golden.pkl will be loaded.
4. Will take around 48 seconds
5. With a pre existing pickle loaded, will take only 5 seconds

Results:

1. Tests on testing data with: **Accuracy:88.04946996466431%**
2. Accuracy for unknown words=0 (since there is no smoothing)

Qn 2)

Ans:

Added smoothing using:

$$P(w \mid t) = \frac{C(w,t)+1}{C(t)+K}$$

This can be seen in lines 163,165, 193 of file initializer.py

Data:

6. Creates and trains a model on the training data
7. Tests on testing data with: Accuracy:92.93286219081273%
8. Accuracy increased
9. Will take around 117 seconds
10. Accuracy for unknown words=67%

Qn3)

Ans:

Didn't get time to attempt viterbi

Qn 4)

Ans:

Steps to run the program:

1. Once inside the conda environment, cd to the directory called q4
2. python initializer.py

Features:

1. Takes around 5 minutes to train
2. Currently set for one epoch only to save time.
3. This can be changed in lstm.py as noofEpochs

4. Will show progress bar
5. Stores the lstm model as a pkl in the same directory
6. Pick run with pickle for faster run time.
7. I print the scores predicted for each word for first sentence in test data against each POS tags in a matrix of size VxD
    a. V= size of vocabulary
    b. D=number of pos tags
8. Didn't get time to calculate accuracy alone. Otherwise trains and tests correctly.

Qn 5)**Ans:**

As of writing of this article, 2011, the **state of the art** in POS tagging was at **97.44%** as shown by Spoustová et al[3]. This was done using a semi- supervised methodology used as an addendum to the supervised averaged perceptron. To elucidate, they feed into the perceptron a mixture of both manually tagged and machine tagged (unsupervised) data. In this work, Manning proposes a methodology based on a Maximum entropy cyclic dependency network (now part of the Stanford POS tagger) which achieves **97.67%.** The features used in this work include using templates, unknown word features, a bi directional cyclic dependency tagger etc. They also mention that the numbers are improved a bit by reducing the support threshold for rare words. They also improve the performance by using 2 words to the left and right of the tagged word, instead of one, along with the usage of equivalence classes and distributional similarity classes while ignoring splitting tags. As of today Nov 2017, as given in [1] the latest in the POS tagging is at 97.64% per token accuracy achieved by Choi et al. [2] which uses Dynamic Feature Induction as tested on the Wall Street Journal corpus of Penn Tree Bank.

Manning suggests a few categories of errors which contribute towards the **challenges** in increasing the performance. One such category is of **lexicon gap** where a particular word had multiple tags, of which all in training data were tag1. However, in testing time, the word was supposed to have tag2, which for some reason never occurred in the training data. Another problem is that of **unknown word** : that is, the tagger does not know how to tag a word it hasn't seen before, but is seeing for the first time in the test data. Other challenges include the sequence tagger not being able to **plausible recognize the tag** of word from the context, **difficult linguistics problem,** where the tagger is supposed to have knowledge from a broader context than just the sentence it is reading. Though at this point I wonder if an LSTM based mechanism has been proposed to solve this, which probably seems like a possible solution for remembering the broad contextual knowledge part. Another category he mentions is that of **unclear** definitions of tags, where despite linguistic knowledge the answer of which tag to apply itself is ambiguous in a given sentence. Another problem he mentions is that of the **annotators** who tagged the penn tree bank **not being consistent** or even being **wrong**. He mentions certain examples in which the same word has been given two different tags in the same context, in two different sentences. After all this he mentions certain **foundational linguistic issues** where a problem with discrete and fuzzy categories is mentioned. Further he points to the fact that the annotators who developed the gold standard probably were assigning tags based on semantic function instead of the true syntactic distributional categories.

As **solutions** to these problems, he suggests certain deterministic rules which were implemented using Tregex. One such error correction is where he replaced the **past tens**e VBD tag with the past participle tag VBN. This was achieved using better rules over tree pattern. Another such error he points out is where the Penn Treebank annotators tag nouns ending in s (Eg: United States) as **plural** (NNPS) instead of the correct NNP **singular** tag. Another similar problem is that of assigning non-nouns as adjectives when it become part of Nouns. He also corrects the tags for the word *that* depending on the function ot be DT, IN , WDT or RB. He also mentions that all these rules are based on the training data, but there might be instances in the testing data where these rules might not completely get the correct tag. He however says that the maximum he could do was to test it on a development set and ensure that the rules he newly created follows the Hippocratic reasoning of doing no harm.

After adding all these methods and solutions, he achieves an accuracy of 97.75% on the development data and 97.67% on the test data.

References:

1. https://aclweb.org/aclwiki/POS_Tagging_(State_of_the_art)
2. Dynamic Feature Induction: The Last Gist to the State-of-the-Art, Jinho D. Choi, Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (NAACL'16), San Diego, CA, 2016.
3. Hajič, Jan, Jan Raab, and Miroslav Spousta. "Semi-supervised training for the averaged perceptron POS

tagger." *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2009.