Homework 4, Part B: Structured perceptron

Aditya Shastry

October 28, 2016

1 Perceptron and Averaged Training

1.1. The computational advantage of using equation 2 over equation 1 is not having to store all thr previous versions of the weight θ. Also, instead of computing all the additions over the different θ states, only one computation is needed to get the average.

1.2. $\bar{\theta}_t = \frac{1}{t}\sum_{t'=1}^{t} \theta_{t'}$
$\bar{\theta}_1 = \theta_{1'} = \theta_{0'} + rg_1$
$\bar{\theta}_2 = \frac{1}{2}(\theta_{1'} + \theta_{2'}) = \frac{1}{2}(2\theta_{1'} + rg_2) = \frac{1}{2}(2\theta_{0'} + r(2g_1 + r_2))$
$\bar{\theta}_3 = \frac{1}{3}(\theta_{1'} + \theta_{2'} + \theta_{3'}) = \frac{1}{3}(\theta_{1'} + 2\theta_{2'} + rg_3) = \frac{1}{3}(3\theta_{1'} + r(2g_2 + g_3)) = \frac{1}{3}(3\theta_{0'} + r(3g_1 + 2g_2 + g_3))$
$\bar{\theta}_4 = \frac{1}{4}(\theta_{1'} + \theta_{2'} + \theta_{3'} + \theta_{4'}) = \frac{1}{4}(\theta_{1'} + \theta_{2'} + 2\theta_{3'} + rg_4) = \frac{1}{4}(\theta_{1'} + 3\theta_{2'} + r(2g_3 + g_4)) = \frac{1}{4}(4\theta_{1'} + r(3g_2 + 2g_3 + g_4)) = \frac{1}{4}(4\theta_{0'} + r(4g_1 + 3g_2 + 2g_3 + g_4))$

1.3. $S_1 = S_0 + (1-1)rg_1 = S_0 = 0$
$S_2 = S_1 + (2-1)rg_2 = 0 + rg_2 = rg_2$
$S_3 = S_2 + (3-1)rg_3 = rg_2 + 2rg_3$
$S_4 = S_3 + (4-1)rg_4 = rg_2 + 2rg_3 + 3rg_4$

1.4. $\bar{\theta}_3 = \theta_3 - \frac{1}{3}S_3 = \theta_3 - \frac{1}{3}(rg_2 + 2rg_3) = \frac{1}{3}(3\theta_3 - rg_2 - 2rg_3) = \frac{1}{3}(3(\theta_2 + rg_2) - rg_2 - 2rg_3)$
$= \frac{1}{3}(3(\theta_1 + rg_1 + rg_2) - rg_2 - 2rg_3) = \frac{1}{3}(3(\theta_0 + rg_1 + rg_2) - rg_2 - 2rg_3) = \frac{1}{3}(3\theta_0 + 3rg_1 + 2rg_2 + rg_3)$
$= \frac{1}{3}(3\theta_0 + r(3g_1 + 2g_2 + g_3))$ which is same as the value for $\bar{\theta}_3$ from 1.2
$\bar{\theta}_4 = \theta_4 - \frac{1}{4}S_4 = \frac{1}{4}(4\theta_4 - rg_2 - 2rg_3 - 3rg_4)$
expanding on θ_4 similar to θ_3 as shown above:
$\bar{\theta}_4 = \frac{1}{4}(4\theta_0 + 4rg_1 + 4rg_2 + 4rg_3 + 4rg_4 - rg_2 - 2rg_3 - 3rg_4) = \frac{1}{4}(4\theta_0 + r(4g_1 + 3g_2 + 2g_3 + g_4)$ which is same as the value for $\bar{\theta}_4$ from 1.2

1.5. Proof by induction for $\bar{\theta}_t = \theta_t - \frac{1}{t}S_t$
Proof for t = 1:
$\bar{\theta}_1 = \theta_1 - \frac{1}{1}S_1 = \theta_1 = \sum_{t=1}^{1} \theta_i$
Assuming truth for t = k:
$\bar{\theta}_k = \theta_k - \frac{1}{k}S_k$
To prove t = k+1: $\bar{\theta}_{k+1} = \theta_{k+1} - \frac{1}{k+1}S_{k+1}$
$\bar{\theta}_{k+1} = \frac{1}{k+1}\sum_{i=1}^{k+1} \theta_i = \frac{1}{k+1}(\theta_{k+1} + \sum_{i=1}^{k} \theta_i) = \frac{1}{k+1}(\theta_{k+1} + \frac{k}{k}\sum_{i=1}^{k} \theta_i)$
after substituting from the assumption of t = k ()
$= \frac{1}{k+1}(\theta_{k+1} + k(\theta_k - \frac{1}{k}S_k)) = \frac{1}{k+1}(\theta_{k+1} + k\theta_k - S_k)$
adding and subtracting krg_{k+1}
$= \frac{1}{k+1}(\theta_{k+1} + k\theta_k - S_k + krg_{k+1} - krg_{k+1}) = \frac{1}{k+1}(\theta_{k+1} + k(\theta_k + rg_{k+1}) - (S_k + krg_{k+1}))$
$= \frac{1}{k+1}(\theta_{k+1} + k\theta_{k+1} - S_{k+1} = \theta_{k+1} - \frac{1}{k+1}S_{k+1}$
This is the proof for t = k+1.

2 Classifier Perceptron

2. The bug is in the statement `diffs[k] += predfeats[k]`.
The issue here is that the corresponding feature values for the predicted feature vector `predfeats` should be subtracted from the respective golden geature vector `goldfeats` to improve the score for the gold feature values and reduce the score for the wrong predicted feature values. Here as the scores for the predicted features are being added to update the weights, this improves the scores of wrong predictions. If this is not fixed and the programmer runs the code as is, the predictions would be randomly made as the scores for both right and wrong predictions are increased. The accuracy of predictions would not improve and weights will never converge to an optimal value to make accurate predictions.

3 Structured Perceptron with Viterbi

3.1. The below is the output of the script:

```
Most common tag is V
Accuracy is 15.5712212316
```

3.2. The below code snippet was used for testing *local_emission_features*:

```
tokens = ["the", "old", "man", "the", "boat"]
tags = ["D", "N", "V", "D", "N"]
for index, tag in enumerate(tags):
print structperc.local_emission_features(index, tag, tokens)
```

The below is the output from the code:

```
'tag=D_biasterm':  1, 'tag=D_curword=the':  1
'tag=N_curword=old':  1, 'tag=N_biasterm':  1
'tag=V_curword=man':  1, 'tag=V_biasterm':  1
'tag=D_biasterm':  1, 'tag=D_curword=the':  1
'tag=N_curword=boat':  1, 'tag=N_biasterm':  1
```

Each feature returns 2 values - the bias term for the label/tag which helps to choose the label/tag when in doubt regarding the transition scores, and the current word term which provides a binary value (1 in case the tag is observed for the token, 0 otherwise).

3.3. The function *features_for_seq* has been implemented in the code.
The below is the output (only the non zero features) for the sentence `the old man the boat` and a tag sequence `["D", "N", "V", "D", "N"]`:

```
('V', 'bias'):  1, ('V', 'D'): 1, ('V', 'man'):  1, ('N', 'boat'):  1, ('D', 'bias'):
2, ('D', 'N'): 2, ('D', 'the'):  2, ('N', 'old'):  1, ('N', 'bias'):  1, ('N', 'V'):
1
```

3.4. The code for *calc_factor_scores* needs to calculate A and B scores. The A scores are the transition scores between different tags/labels. This can be taken directly from the weights. The B scores are the emission scores which need the observation, bias and other features for a tag/label. These can be obtained from function *local_emission_features* and multiplied with the respective weight feature values to obtain the B score for the tag/label. The function *features_for_seq* calculates the count for the transitions only in the provided sequence, and calculates the B scores only based on the count of bias, observation and other features. As the weights parameter doesnt influence the scores, the scores only for a single sequence is calculated and not for all possible sequences out of which the best can be chosen by viterbi algorithm. So, the function *features_for_seq* should not be called.

3.5. The function *calc_factor_scores* has been implemented in the code.
The call:

```
tokens = ["the", "old", "man", "the", "boat"]
perc_weights = ("D", "N"):  20, ("D", "V"):  0, ("N", "V"):  10, ("N", "D"):  0,
("V", "D"):  10, ("V", "N"):  0
print structperc.calc_factor_scores(tokens, perc_weights)
```

The output:

```
(('N', 'D'): 0, ('N', 'V'): 10, ('V', 'V'): 0.0, ('D', 'N'): 20, ('V', 'N'): 0, ('V',
'D'): 10, ('D', 'V'): 0, ('N', 'N'): 0.0, ('D', 'D'): 0.0, ['V': 0.0, 'D': 0.0, 'N':
0.0, 'V': 0.0, 'D': 0.0, 'N': 0.0, 'V': 0.0, 'D': 0.0, 'N': 0.0, 'V': 0.0, 'D': 0.0,
'N': 0.0, 'V': 0.0, 'D': 0.0, 'N': 0.0])
```

3.6. The function *predict_seq* has been implemented in the code. It calls the *calc_factor_scores* function to get the A and B scores from the weights and tokens. Then it calls the viterbi function submitted as part of hw4_a.

3.7. The function *train* has been implemented in the code. It produces the below output:

```
Training iteration 0
TR RAW EVAL: 8703/14619 = 0.5953 accuracy
DEV RAW EVAL: 2611/4823 = 0.5414 accuracy
DEV AVG EVAL: 2898/4823 = 0.6009 accuracy
Training iteration 1
```

```
TR RAW EVAL: 9975/14619 = 0.6823 accuracy
DEV RAW EVAL: 2775/4823 = 0.5754 accuracy
DEV AVG EVAL: 3041/4823 = 0.6305 accuracy
Training iteration 2
TR RAW EVAL: 10199/14619 = 0.6977 accuracy
DEV RAW EVAL: 2808/4823 = 0.5822 accuracy
DEV AVG EVAL: 3016/4823 = 0.6253 accuracy
Training iteration 3
TR RAW EVAL: 9587/14619 = 0.6558 accuracy
DEV RAW EVAL: 2682/4823 = 0.5561 accuracy
DEV AVG EVAL: 2993/4823 = 0.6206 accuracy
Training iteration 4
TR RAW EVAL: 9636/14619 = 0.6591 accuracy
DEV RAW EVAL: 2736/4823 = 0.5673 accuracy
DEV AVG EVAL: 3038/4823 = 0.6299 accuracy
Training iteration 5
TR RAW EVAL: 10830/14619 = 0.7408 accuracy
DEV RAW EVAL: 2902/4823 = 0.6017 accuracy
DEV AVG EVAL: 3033/4823 = 0.6289 accuracy
Training iteration 6
TR RAW EVAL: 10139/14619 = 0.6935 accuracy
DEV RAW EVAL: 2748/4823 = 0.5698 accuracy
DEV AVG EVAL: 3045/4823 = 0.6313 accuracy
Training iteration 7
TR RAW EVAL: 10352/14619 = 0.7081 accuracy
DEV RAW EVAL: 2766/4823 = 0.5735 accuracy
DEV AVG EVAL: 3007/4823 = 0.6235 accuracy
Training iteration 8
TR RAW EVAL: 10212/14619 = 0.6985 accuracy
DEV RAW EVAL: 2812/4823 = 0.5830 accuracy
DEV AVG EVAL: 2980/4823 = 0.6179 accuracy
Training iteration 9
TR RAW EVAL: 10294/14619 = 0.7042 accuracy
DEV RAW EVAL: 2771/4823 = 0.5745 accuracy
DEV AVG EVAL: 3021/4823 = 0.6264 accuracy
Learned weights for 23402 features from 1000 examples
```

3.8. Accuracy rate for each tag in development set, using the code fancy_eval, is as below:

```
gold P acc 0.8977 (395/440)
gold D acc 0.8365 (261/312)
gold O acc 0.8108 (270/333)
gold & acc 0.7912 (72/91)
gold , acc 0.7760 (388/500)
gold N acc 0.7758 (512/660)
gold V acc 0.6178 (464/751)
gold !  acc 0.4949 (49/99)
gold L acc 0.4923 (32/65)
gold R acc 0.4402 (92/209)
gold A acc 0.3682 (88/239)
gold E acc 0.3462 (18/52)
gold $ acc 0.3372 (29/86)
gold T acc 0.3333 (12/36)
gold G acc 0.2154 (14/65)
gold ^ acc 0.1961 (61/311)
gold # acc 0.0577 (3/52)
gold @ acc 0.0412 (10/243)
gold U acc 0.0110 (1/91)
gold S acc 0.0000 (0/5)
```

```
gold X acc 0.0000 (0/4)
gold Z acc 0.0000 (0/9)
gold   acc 0.0000 (0/170)
```
For 2 sample sentences from the dev set, below is the comparison between gold labels and predicted labels:
```
@ciaranyree it was on football wives , one of the players and his wife own smash
burger
@', 'O', 'V', 'P', 'N', 'N', ',', '$', 'P', 'D', 'N', '', 'D', 'N', 'V', '^', '^'
,', 'O', 'V', 'P', 'N', 'N', 'P', '$', 'P', 'D', 'N', '', 'D', 'N', 'N', 'N', ','
RT @TheRealQuailman :  Currently laughing at Laker haters .
 ', '@', ' ', 'R', 'V', 'P', '^', 'N', ','
N', 'N', 'N', 'N', 'N', 'P', '^', 'N', ','
```
By the look of the above examples, the common errors are made for identifying twitter tags (words beginning with @). Some of the predictions made for them are nouns, maybe due to higher bias for them. Another noticable error is marking words ending with -ing as nouns, instead of the correct verbs. This might be due to more gold labels marking words ending with -ing as nouns (eg. participles and gerunds). While there are some errors, the accuracies for some of the common POS tags/labels, (like determiner, noun, and verb) are high