Homework 4 Part B: Structured Perceptron

Shamya Karumbaiah [Collaborated with - Rafael Lizarralde]

October 27, 2016

1 Perceptron and Averaged Training

1.1

Equation 1 needs storing all the $\theta_{t'}$ for $t' = 1, ..t$ to calculate $\bar{\theta_t}$ and needs averaging over all these values. The number of $\theta_{t'}$ to be stored and computed upon is equal to the product of number of training examples and number of passes. With equation 2 we could avoid this and compute $\bar{\theta_t}$ just with the current values of θ_t and S_t.

1.2

Using equation 1, assuming $\theta_0 = 0$ we have
$\bar{\theta_1} = \theta_1 = rg_1$
$\bar{\theta_2} = \frac{\theta_1+\theta_2}{2} = \frac{2rg_1+rg2}{2}$
$\bar{\theta_3} = \frac{\theta_1+\theta_2+\theta_3}{3} = \frac{3rg_1+2rg_2+rg_3}{3}$
$\bar{\theta_4} = \frac{\theta_1+\theta_2+\theta_3+\theta_4}{4} = \frac{4rg_1+3rg_2+2rg_3+rg_4}{4}$

1.3

Using weightsums definition, assuming $S_0 = 0$, we have
$S_1 = 0$
$S_2 = rg_2$
$S_3 = rg_2 + 2rg_3$
$S_4 = rg_2 + 2rg_3 + 3rg_4$

1.4

Using equation 2 definition, assuming $\theta_0 = 0, S_0 = 0$, we have
$\bar{\theta_3} = \theta_3 - \frac{S_3}{3} = (rg_1 + rg_2 + rg_3) - (\frac{rg_2+2rg_3}{3}) = \frac{3rg_1+2rg_2+rg_3}{3}$
$\bar{\theta_4} = \theta_4 - \frac{S_4}{4} = (rg_1 + rg_2 + rg_3 + rg_4) - (\frac{rg_2+2rg_3+3rg_4}{4}) = \frac{4rg_1+3rg_2+2rg_3+rg_4}{4}$
These are same as the values computed in 1.2

2 Classifier Perceptron

The problem in the code is at **diffs [k] += predfeats [k]**. This should have been **diffs [k] -= predfeats [k]**. Subtraction ensures that when the predicted label is equal to the true label, the weights are left unchanged as the difference would result in a zero vector. The weights are only changed when the model gets its prediction wrong during which the features only present in the predicted set and not in true set are decrement to have less weightage. If the code is run unchanged, the weights would keep wandering without stabilizing and will end up in a random weight whose predictions would be random as well. If the features can take only positive values, there is a chance for the model to become a constant predictor.

3 Structured Perceptron with Viterbi

3.1

Code - *dataanalysis.py*
most common tag - V
accuracy if we predict it for all tags: 15.57%

3.2

Ex 1. function call - *local_emission_features(1,'V', ['I','love','cats'])*
output - {*'tag=V_curword=love': 1, 'tag=V_biasterm': 1*}

Ex 2. function call - *local_emission_features(0,'A', ['happy','president','Abe'])*
output - {*'tag=A_biasterm': 1, 'tag=A_curword=happy': 1*}

On calling local_emission_features with a tag and position(t), it gives the local observation feature which is nothing but $f(y_t, x_t)$ set to 1 for the token x_t at position t and tag y_t. In ex 1 it is $f(V, love) = 1$ and in ex 2 it is $f(A, happy) = 1$.

3.3

Function call - *features_for_seq(['Happy','president','Abe', 'Lincoln'], ['A','N','N','N'])*
output - {*'tag=A_biasterm': 1, 'tag=A_curword=Happy': 1, 'tag=N_curword=Abe': 1, 'lasttag=A_curtag=N': 1, 'tag=N_curword=president': 1, 'lasttag=N_curtag=N': 2, 'tag=N_biasterm': 3, 'tag=N_curword=Lincoln': 1*}

3.4

Local feature decomposition implies that the full feature vector can be obtained just by summing up all the local features. Similarly, with scoring function decomposition, we could calculate the full score function just summing the local products of the local features and corresponding weights. So, instead of generating each combination of label sequence and calling the features_for_seq, it is enough to call the local_emission_features for each tag and token locally.

3.5

Please note, the vocabulary was trimmed for this exercise, OUTPUT_VOCAB = (V, P)
Setting some random weights,
weights['tag=V_curword=love'] = 1.6
weights['tag=P_curword=I'] = 0.9
weights['lasttag=V_curtag=P'] = 1.2
weights['lasttag=P_curtag=V'] = 0.2
weights['tag=V_biasterm'] = 0.8
weights['tag=P_biasterm'] = 1.4

Function call - *A, B = calc_factor_scores(['I','love'], weights)*
Output - {('V', 'V'): 0.0, ('V', 'P'): 1.2, ('P', 'P'): 0.0, ('P', 'V'): 0.2}
$[\{'P' : 2.3, 'V' : 0.8\}, \{'P' : 1.4, 'V' : 2.4000000000000004\}]$

3.6

Nothing to report. I am using vit.viterbi() for decoding.

3.7

Output as expected -

Training iteration 0
TR RAW EVAL: 8523/14619 = 0.5830 accuracy
DEV RAW EVAL: 2556/4823 = 0.5300 accuracy
DEV AVG EVAL: 2986/4823 = 0.6191 accuracy
Training iteration 1
TR RAW EVAL: 10643/14619 = 0.7280 accuracy
DEV RAW EVAL: 2956/4823 = 0.6129 accuracy
DEV AVG EVAL: 3170/4823 = 0.6573 accuracy
Training iteration 2
TR RAW EVAL: 10148/14619 = 0.6942 accuracy
DEV RAW EVAL: 2692/4823 = 0.5582 accuracy
DEV AVG EVAL: 3281/4823 = 0.6803 accuracy
Training iteration 3
TR RAW EVAL: 11681/14619 = 0.7990 accuracy
DEV RAW EVAL: 3150/4823 = 0.6531 accuracy
DEV AVG EVAL: 3311/4823 = 0.6865 accuracy
Training iteration 4
TR RAW EVAL: 11722/14619 = 0.8018 accuracy
DEV RAW EVAL: 3028/4823 = 0.6278 accuracy
DEV AVG EVAL: 3322/4823 = 0.6888 accuracy
Training iteration 5
TR RAW EVAL: 11799/14619 = 0.8071 accuracy
DEV RAW EVAL: 3003/4823 = 0.6226 accuracy
DEV AVG EVAL: 3333/4823 = 0.6911 accuracy
Training iteration 6

TR RAW EVAL: 10765/14619 = 0.7364 accuracy
DEV RAW EVAL: 2839/4823 = 0.5886 accuracy
DEV AVG EVAL: 3340/4823 = 0.6925 accuracy
Training iteration 7
TR RAW EVAL: 12355/14619 = 0.8451 accuracy
DEV RAW EVAL: 3076/4823 = 0.6378 accuracy
DEV AVG EVAL: 3346/4823 = 0.6938 accuracy
Training iteration 8
TR RAW EVAL: 11224/14619 = 0.7678 accuracy
DEV RAW EVAL: 2947/4823 = 0.6110 accuracy
DEV AVG EVAL: 3349/4823 = 0.6944 accuracy
Training iteration 9
TR RAW EVAL: 12581/14619 = 0.8606 accuracy
DEV RAW EVAL: 3232/4823 = 0.6701 accuracy
DEV AVG EVAL: 3341/4823 = 0.6927 accuracy

3.8

Output of fancy_eval, accuracy of each tag in dev data set -
gold O acc 0.9489 (316/333)
gold , acc 0.9480 (474/500)
gold D acc 0.9295 (290/312)
gold & acc 0.9231 (84/91)
gold P acc 0.9205 (405/440)
gold V acc 0.8322 (625/751)
gold L acc 0.7538 (49/65)
gold T acc 0.7500 (27/36)
gold N acc 0.7000 (462/660)
gold R acc 0.6555 (137/209)
gold A acc 0.5900 (141/239)
gold E acc 0.5577 (29/52)
gold ! acc 0.5253 (52/99)
gold @ acc 0.4280 (104/243)
gold $ acc 0.3605 (31/86)
gold U acc 0.2747 (25/91)
gold ^ acc 0.2379 (74/311)
gold G acc 0.2000 (13/65)
gold # acc 0.0577 (3/52)
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 the first tweet in devdata using the final weight after model training, the predictions are as below -

word	gold	pred
@ciaranyree	@	@

it	O	O	
was	V	V	
on	P	P	
football	N	N	
wives	N	V	*** Error
,	,	,	
one	$	$	
of	P	P	
the	D	D	
players	N	N	
and	&	&	
his	D	D	
wife	N	N	
own	V	N	*** Error
smash	^	V	*** Error
burger	^	@	*** Error

For the second tweet it is,

word	gold	pred	
RT	~	A	*** Error
@TheRealQuailman	@	N	*** Error
:	~	,	*** Error
Currently	R	A	*** Error
laughing	V	N	*** Error
at	P	P	
Laker	^	^	
haters	N	N	
.	,	,	

Consulting the tagset description at *Gimpel et al. 2011*, it is evident that the tags with lowest accuracies have the lowest relative frequency in the data like S (0.1), X(0.1) and Z(0.2). The model is able to identify the tags like O,D,P,V and punctuation with good accuracy. These tags have common words that occur frequently in a text unlike say N which can take varied values adding a lot of ouliers.

3.9

System	Number of weight values	Accuracy
basic	24361	0.6927
basic plus first 2 characters RT	24340	0.6942
basic plus first character #	23761	0.6998
basic plus URL detector	23580	0.7085
basic plus first character @	21517	0.7487
With all 4 features	20447	**0.7622**

Below is the tag wise accuracy for the final model with all features. Note the significant improvement in the accuracies of the tags worked on -

gold @ acc 1.0000 (243/243)
gold , acc 0.9660 (483/500)
gold # acc 0.9615 (50/52)
gold & acc 0.9560 (87/91)
gold O acc 0.9550 (318/333)
gold D acc 0.9327 (291/312)
gold P acc 0.9295 (409/440)
gold U acc 0.9231 (84/91)
gold V acc 0.8522 (640/751)
gold T acc 0.7778 (28/36)
gold N acc 0.7773 (513/660)
gold L acc 0.7692 (50/65)
gold R acc 0.6842 (143/209)
gold A acc 0.5774 (138/239)
gold ! acc 0.5455 (54/99)
gold E acc 0.5192 (27/52)
gold $ acc 0.3605 (31/86)
gold G acc 0.2615 (17/65)
gold ˆ acc 0.2251 (70/311)
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)