Independent Project 2: Sequence Labeling

CS 6501-005 Natural Language Processing

Murugesan Ramakrishnan - mr6rx

1 Hidden Markov Model

1.1 A Toy Problem

Note: Since the Viterbi algorithm works with log scale, all the input transition and emission probabilities have been convereted to log space

1. Trellis Table - vm values

	G	C	A	C	T	G	END
H	-1.714798	-3.835062	-6.360791	-8.481054	-11.006783	-12.968441	-13.66
L	-2.525729	-4.240527	-5.955326	-8.257911	-10.155031	-12.457616	

   Trellis Table - bm values

G	C	A	C	T	G	END
START	H	H	H	H	L	L
START	H	H	L	L	L	

2. Hidden state of the DNA sequence GCACTG is HHLLLL

1.2 POS Tagging

1. Preprocessing: K value chosen as 10. Words with frequency less than or equal 10 have been mapped to 'Unk'. Based on this K, we get the Vocab size (unique list of words) as 7134

2. The transition probability has been estimated and stored in *mr6rx-tprob.txt* file. Total number of records in the file are 121

3. The emission probability has been estimated and stored in *mr6rx-eprob.txt* file. Total number of records in the file are 71340

4. Adding values $\alpha = 1$ and $\beta = 1$ to the MLE. Emission and Transition probabilities calculated as specified and stored in variables *emission_prob_df_alpha* and *transition_prob_beta_df*. Files mr6rx-eprob-smoothed.txt and mr6rx-tprob-smoothed.txt created with re-estimated probabilites.

5. Log space and Viterbi: Ran the Viterbi algorithm with log probabilites and achieved a tag level accuracy of 93.88% for the dev dataset

6. Ran decoder on test data and the file is saved as *mr6rx-viterbi.txt*

7. Tuning values of α and β to get the best prediction performance. Following are the various values of alpha and beta tested for K = 10

`alpha`	`beta`	Dev Accuracy(%)
0.1	0.1	93.9
0.5	0.5	93.83
1	1	93.88
2	1	93.40
3	1	93.05
3	2	93.05
3	3	93.88

   Based on these different values, final values are selected as α = 0.1 and β = 0.1.

8. Based on the final values of α and β the emission and transition probabilities are recalculated and converted to log space which is then used for predicting the tags in the test dataset.The file is stored as *mr6rx-viterbi-tuned.txt*

2 Conditional Random Field

1. Augmenting the model with more features: Following are the features added,
   i) First letter of the word
   ii) Last letter of the word
   iii) First two letters of the word
   iv) Last two letter of the word
   v) Previous word
   vi) Next word
   vii) First character upper - 1 if first character is upper 0 otherwise
   viii) Entire word upper - 1 if all characters are upper 0 otherwise
   ix) First three letters of the word
   x) Last three letters of the word

   Model Performance,
   Training set Accuracy: 71.65%
   Dev set Accuracy: 71.66%

2. * Switching the value of algorithm from *lbfgs* to *averaged perceptron.* The accuracy is as follows,
   Training set Accuracy: 97.33%
   Dev set Accuracy: 87.26%

   * USAGE OF AVERAGED PERCEPTRON FOR CRF

   While in logistic Regression we try to find the value of $\widehat{y}$ by maximizing the probability function P(y′|x;θ), in average perceptron $\widehat{y}$ is determined by finding the optimal θ such that we maximize $\theta^{\top} f(x, y\prime)$ where θ is the weights for each of the input sequence (Features for the model in this case - First letter, First two letters, previous work and so on). CRF uses this estimate of $\widehat{y}$.

For a series of input sequence X, we create all possible feature functions f(X,y), i.e. say f(X,y1) f(X,y2) and f(X, y3) given 3 values of y. Now, with these as the feature function, the perceptron algorithm is implemented to get $y\prime$ by getting a converged value of θ by iterating over multiple epochs. This portion of the algorithm is the difference between CRF with logistic and CRF with averaged perceptron.

Note: Certain changes to the existing functions were made to accomodate with Python3. The *print* functions were modified, *encoding = 'utf-8'* was specified while reading the file and \n was added to line.strip(\n)