Assignment 2

Viterbi Implementation

The Viterbi algorithm is an extension of minimum edit distance which uses probabilistic definitions of the operations. The Viterbi algorithm takes a set of observed words as input and returns the most probable state/tag sequence together with its probability.

In my Viterbi implementation, I have used two lists, y and path, where the list y is storing the maximum score up to a point and a list path indicating the path taken up to that point. First, the list y is populated with the starting scores of the tags passed to the function and the emission scores of the first element. Then we loop from the second word to the end of the sentence. In each iteration, we compute the value of y by adding the previous score, the emission scores and the transition scores. We get the score having the highest probability for the word and then iterate further with the help of this score to compute for the other words. Finally, we add the end scores for each tag to the last column of y scores to find the maximum score of the word sequence and get its corresponding path. We return this in the function response.

CHALLENGES FACED:

I was trying to do multiplication previously as I did not notice that log values have been provided for the scores, so was not getting accurate results. But then after going through the FAQs, I got to know this and did additions instead. Also, the FAQs helped me to figure out what's the use of end scores in this implementation.

Added Features

I have implemented the below sets of features to test the token accuracy.

1. **Number of characters**: To check the length of the word and determine whether it has any affect in the token accuracy.
2. **Is Capitalized**: To check whether the first word is capitalized or not.

3. **Capitals Inside**: Checking whether the word has got any capital letters in the middle.
4. **Prefix and Suffix**: To extract the first three and last three characters of the word and check whether it changes the token frequency.
5. **Punctuation**: Checking whether the word has got any punctuation characters like ‘,’,’- ‘,’?’,’ ()’, etc
6. **Hyperlink**: To check whether the word has hyperlink characters like ‘@’,’http’.
7. **Starts with special character**: Checking whether the word starts with a special character like ‘#’,’@’, as we see in many tweets.
8. **Stop Words**: With the help of NLTK, I could add these features to check whether the word is a stopword or not.
9. **Emoticon**: Tweets often contain emojis and it becomes an important feature of the word.

10. **Brown Clustering**: - It is a hierarchical clustering model which assigns all the words in the training set to different clusters and denotes them as bit strings, where each bit string represents the length from the root of the hierarchy tree to the cluster the word is assigned to. The code used was from https://github.com/mheilman/tan-clustering/blob/master/class_lm_cluster.py. The code was only run once to generate the output file, which contained the words, their bit string and the frequency of those words. Post that the file was read to predict the word frequency and to check whether the word is present in the file. If its present we add the first n bit strings of the word to the list ftrs. Here n is the no of digits to add and we can change its value as per requirement.

11. **Word Frequency**: To check the frequency of words appearing in a sentence using the file created by brown clustering

12. **Word Stemming**: - We use the word stemming function from the NLTK library, specifically the SnowballStemmer. We initialize the stemmer and add the stem of each word as a feature.

13. **Named Entity recognition**: For this I used the Stanford NER to extract domain specific information of the word and check to which group does a specific word belong to.

Feature Accuracy with Logistic regression

Feature	Token level accuracy	Sentence level accuracy
All features (1-8)	87.46	14.28
All features (1-9)	87.51	14.28
Word Stemming with all above features	87.65	14.28
Brown Clustering without word stemming	86.32	13.39
Named Entity Recognition	87.7	15.17
Word Stemming with Brown Clustering(n=7)	86.89	14.28
Word Stemming with Brown Clustering(n=15)	86.99	15.17
Brown Clustering without Word Stemming and without word frequency(n=15)	87.22	16.07
Brown Clustering without Word Stemming and without word frequency(n=35)	87.51	16.07
Word Stemming with Brown Clustering and without word frequency(n=35)	87.74	14.28

Feature Accuracy with CRF

Feature	Token level accuracy	Sentence level accuracy
All features (1-8)	86.66	14.28
All features (1-9)	86.66	15.17
Word Stemming with all above features	86.14	14.28
Brown Clustering without word stemming	86.14	13.39
Named Entity Recognition	86.37	16.96
Word Stemming with Brown Clustering(n=7)	86.51	15.17

Brown Clustering with Word Stemming and without word frequency(n=35)	86.99	16.07
Brown Clustering without word stemming and without word frequency(n=35)	85.71	12.5

From the above table we can see that the best performance in both cases is obtained with word stemming and for n=35.

Also, I tried tuning the hyperparameters to check the feature accuracy for the CRF model. Below is the table showing the results:

Number of Iterations:

Number of iterations	Token level accuracy	Sentence level accuracy
30	86.84	15.17
40	86.89	15.17
45	86.70	13.39
50	86.56	13.39

Initially, by increasing the number of iterations the accuracy was increasing, but then the accuracy started decreasing after a specific value.

Batch Learning: By setting the value of batch learning true, the accuracy decreased remarkably.

Average: The accuracy decreased when I changed the value to False.

Comparison against basic features

The token level accuracy was around 84 for both the models on the basic features.

On adding the initial enhanced features (1-8), the token level accuracy and sentence level accuracy did improve for both the models

By adding the emoticon feature the token level and sentence level accuracy showed a significant change in the token and sentence level accuracy.

The word stemming improved sentence level accuracy slightly but had a greater impact on the token level accuracy

With brown clustering both the token level and sentence level accuracy showed a remarkable improvement. By increasing the value of n, the token accuracy and the sentence level accuracy changed significantly.

Named Entity recognition had a greater impact on the sentence level accuracy. Nonetheless the token level accuracy also increased in this case.

Comparison between LR and CRF

Adding just the initial features in both the models LR and CRF, the token level accuracy and the sentence level accuracy increased for both.

By adding the emoticon feature, I could see that the token level accuracy increased for LR, but the sentence level accuracy remained constant. On the other hand, CRF demonstrated a complete different behaviour. The sentence level accuracy increased for CRF, but the token level accuracy remained the same.

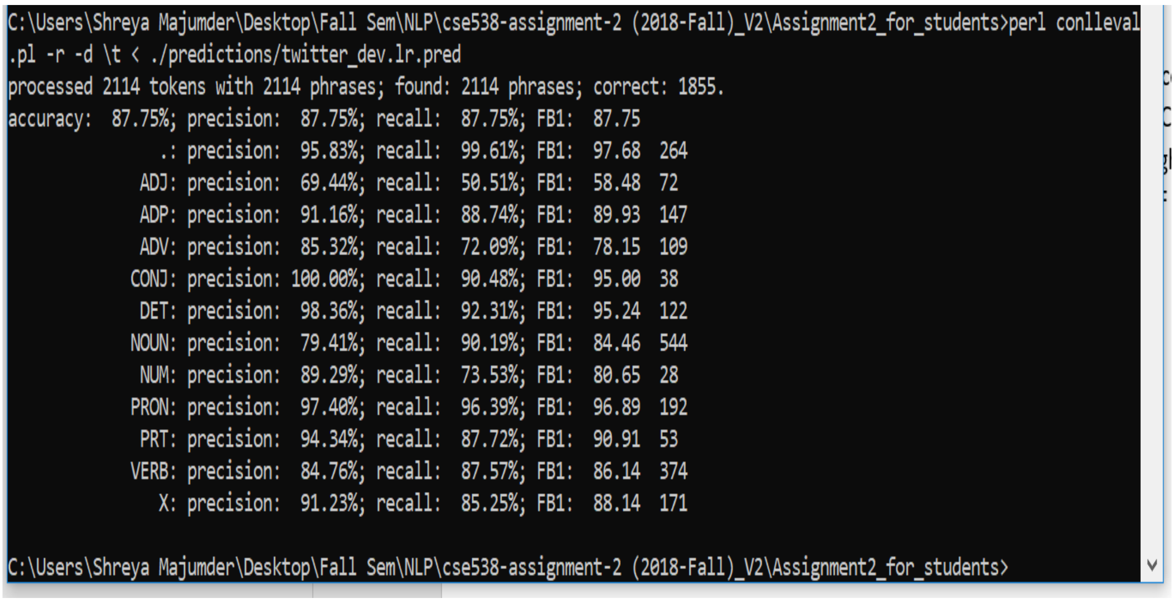
Next, I implemented the word stemming feature. Here also I could see a very contrasting behaviour. While the token accuracy increased remarkably for LR with word stemming, it decreased for CRF.

I also tried brown clustering feature to check the accuracy for the models. The accuracy for the model LR increased by increasing the value of n for brown clustering. Here CRF also depicted a similar behaviour.

Lastly, I tried my hands on implementing Named Entity Recognition. The token accuracy and the sentence level accuracy increased for LR. While for CRF, the token level accuracy decreased but the sentence level accuracy increased significantly.

Comparing logistic regression and CRF for different POS tags, we get the following data using conlleval.pl

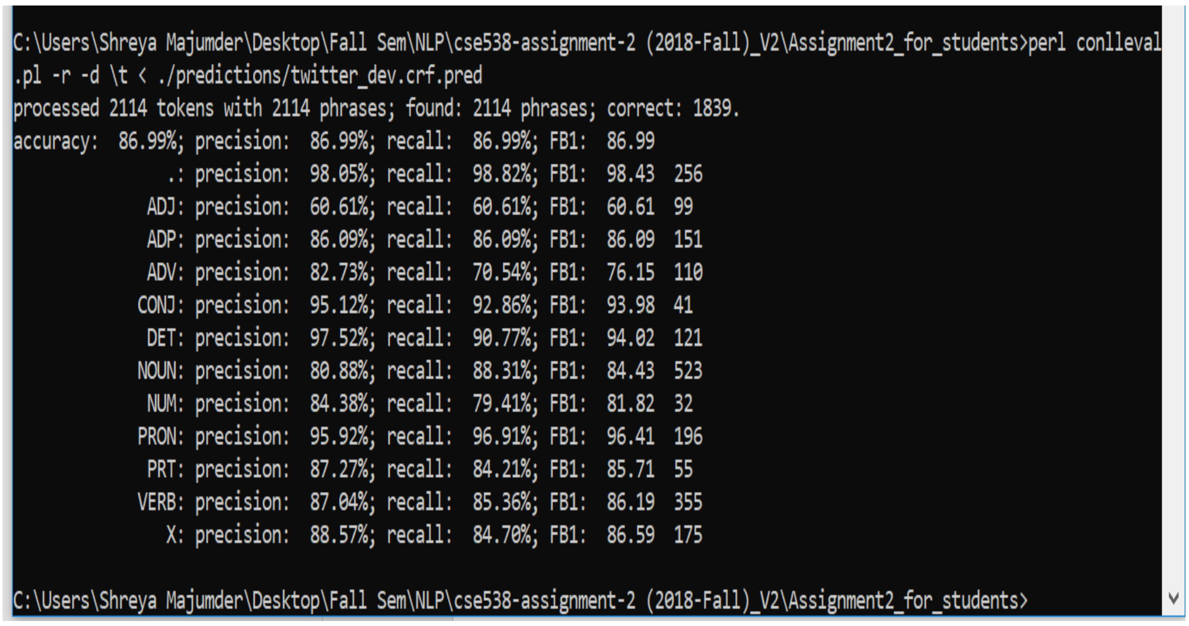
LR:



```
C:\Users\Shreya Majumder\Desktop\Fall Sem\NLP\cse538-assignment-2 (2018-Fall)_V2\Assignment2_for_students>perl conlleval
.pl -r -d \t < ./predictions/twitter_dev.lr.pred
processed 2114 tokens with 2114 phrases; found: 2114 phrases; correct: 1855.
accuracy:  87.75%; precision:  87.75%; recall:  87.75%; FB1:  87.75
                .: precision:  95.83%; recall:  99.61%; FB1:  97.68  264
              ADJ: precision:  69.44%; recall:  50.51%; FB1:  58.48  72
              ADP: precision:  91.16%; recall:  88.74%; FB1:  89.93  147
              ADV: precision:  85.32%; recall:  72.09%; FB1:  78.15  109
             CONJ: precision: 100.00%; recall:  90.48%; FB1:  95.00  38
              DET: precision:  98.36%; recall:  92.31%; FB1:  95.24  122
             NOUN: precision:  79.41%; recall:  90.19%; FB1:  84.46  544
              NUM: precision:  89.29%; recall:  73.53%; FB1:  80.65  28
             PRON: precision:  97.40%; recall:  96.39%; FB1:  96.89  192
              PRT: precision:  94.34%; recall:  87.72%; FB1:  90.91  53
             VERB: precision:  84.76%; recall:  87.57%; FB1:  86.14  374
                X: precision:  91.23%; recall:  85.25%; FB1:  88.14  171

C:\Users\Shreya Majumder\Desktop\Fall Sem\NLP\cse538-assignment-2 (2018-Fall)_V2\Assignment2_for_students>
```

CRF:



```
C:\Users\Shreya Majumder\Desktop\Fall Sem\NLP\cse538-assignment-2 (2018-Fall)_V2\Assignment2_for_students>perl conlleval
.pl -r -d \t < ./predictions/twitter_dev.crf.pred
processed 2114 tokens with 2114 phrases; found: 2114 phrases; correct: 1839.
accuracy:  86.99%; precision:  86.99%; recall:  86.99%; FB1:  86.99
                .: precision:  98.05%; recall:  98.82%; FB1:  98.43  256
              ADJ: precision:  60.61%; recall:  60.61%; FB1:  60.61  99
              ADP: precision:  86.09%; recall:  86.09%; FB1:  86.09  151
              ADV: precision:  82.73%; recall:  70.54%; FB1:  76.15  110
             CONJ: precision:  95.12%; recall:  92.86%; FB1:  93.98  41
              DET: precision:  97.52%; recall:  90.77%; FB1:  94.02  121
             NOUN: precision:  80.88%; recall:  88.31%; FB1:  84.43  523
              NUM: precision:  84.38%; recall:  79.41%; FB1:  81.82  32
             PRON: precision:  95.92%; recall:  96.91%; FB1:  96.41  196
              PRT: precision:  87.27%; recall:  84.21%; FB1:  85.71  55
             VERB: precision:  87.04%; recall:  85.36%; FB1:  86.19  355
                X: precision:  88.57%; recall:  84.70%; FB1:  86.59  175

C:\Users\Shreya Majumder\Desktop\Fall Sem\NLP\cse538-assignment-2 (2018-Fall)_V2\Assignment2_for_students>
```

CRF is better for adjectives, numerals, nouns and verbs. For example, sentences like "At night I 'm wide awake but the daytime, I 'm so tired my day drags", "Five Broken Copy Protection Schemes" has a higher accuracy for the CRF tagger since most elements has a higher accuracy in the CRF tagger then the LR tagger.