The task was to implement a Structured Perceptron which would act as a Named Entity Recogniser (NER). For each word in a given sentence the NER should be able to predict the label for that word.

In the previous lab assignment, the method implemented iterated over all possible Word/Tag Label combinations which although it worked and returned some good results was highly inefficient. The space to iterate over using this method scales by $|y|^{\mathcal{N}}$ so to predict tag labels for longer sentences is unfeasible. Therefore, a new method was required that reduces the number of possible combinations to iterate over with the aim of reducing computation time. This assignment implements the Viterbi Algorithm and Beam Search to reduce the search domain.

The Viterbi Algorithm works by finding the most likely sequence of tags, each tag is determined by splitting the full domain we need to search over into smaller chunks where we are only interested in the likelihood of the labels for the previous chunk of the domain. Doing this means we can assign back-pointers, which are indexes that represent the location of the most probable Word/Tag Label combination in each domain chunk. These indexes create a path so that we can easily find the most likely Word/Tag Label sequence when the full domain has been searched, as we just follow the back-pointer path.

The Beam Search Algorithm is an add on to Viterbi, the depth of the Viterbi matrix could grow large and this would increase the time to find the most probable Word/Tag Label combination in each domain chunk. To stop the matrix from growing too deep, a limit is added that restricts each chuck to a pre-defined maximum depth, whereby only a defined number of the most probable Word/Tag Label combinations remain. The reduced size Beam Search matrix further increases the speed to find most likely Word/Tag Label sequence as there is an even smaller domain to search over.

Code Implementation Explanation

Execution

The file is executed in the terminal using, *python3 lab4.py train.txt test.txt.* Where train.txt stores the training dataset and test.txt stores the testing dataset, both datasets contain pre-processed sentences with assigned entity labels.

Algorithm Implementation

The Viterbi Algorithm is implemented in the *Perceptron* class under the *viterbiMatrix* function. A sentence is supplied to the function along with the weights and currentWord_currentTag count for the training dataset. The sentence is then split into a list of words and tags, for each word every possible combination of word/ Tag is produced. A feature representation is built these combinations using the *phi_1* function, the current term in the Viterbi matrix is compared to the phi_1 feature representation to see whether the current term exists as a feature. If the current term does exist then the feature is multiplied by the features weight, if the current term does not exist as a feature, then the *sumScore* is 0. Next for each current term all the scores from the previous column of the Viterbi matrix are summed and added to the *sumScore,* this score is then added to correct location in the matrix. The back pointer for each location in the matrix is found by using argmax to locate the index of the highest score in the previous column in the matrix. Once the Viterbi matrix has been filled, the highest score on final column of the matrix is found and then using the back pointers the most likely Word/Tag Label sequence for the supplied sentence is produced.

Perceptron / Training

There are two methods to train the Perceptron, *trainPerceptronViterbi* which takes a training dataset and calculates the weights using the plain Viterbi algorithm and the *trainPerceptronBeamSearch* which is very similar to the prior but adds in Beam Search where the Beam Width is pre-determined and entered as an argument to the function. If the predicted Word/ Tag Label sequence produced by each method is incorrect then the weights are updated accordingly. Tags that are false positives have some weight removed and tags that are false negatives have weight added. After each sentence has been iterated through the two sets of weights are returned and then can be used to predict the Word/ Tag Label sequence for unseen data.

Results

Computer specs: MacMini 2018, macOS Version 10.14.3, 6-Core 3.2GHz Intel Core i7-8, 32GB 2667MHz DDR4 RAM.

Perceptron Algorithm	**Average Speed per Epoch (seconds)**	**Time to reach convergence (seconds)**	**F1 Score**
Standard Perceptron	12.51	37.89	0.87034
Perceptron with Viterbi	0.14	0.57	0.87094
Viterbi with Beam Search (Width of 1)	0.09	0.27	0.88092
Viterbi with Beam Search (Width of 2)	0.1	0.30	0.88092
Viterbi with Beam Search (Width of 12)	0.11	0.33	0.88092

Table 1: Comparison of the time to train and test the three implemented Perceptron Models. The Viterbi with Beam Search has been executed with 3 various beam widths to show the effect of beam width on computation time and prediction accuracy. All tests used 10 iterations and the 'O' label was included in model evaluation.

As shown in Table 1, the time taken to complete a training epoch in the perceptron with Viterbi is just over 1% of the time taken to complete an epoch in the standard perceptron. This difference in speed will only increase if sentences were used that had more words. The sentences used in this assignment had few words, with an average sentence length of less than 5. Most sentences in the English language contain far more words than this number, this would exponentially increase the computation time for standard perceptron whilst the increase in computation time for the Viterbi perceptron would remain relatively linear.

Beam search does not appear to alter the accuracy of the predictions, there is only difference of 0.00998 in the F1 scores. This is probably because there are not many tags used in this assignment, so the Viterbi matrix is never capable of becoming very deep. Therefore, when the matrix depth is restricted, only a small amount of the search domain is lost. The effect on accuracy when changing the Beam Search width is also dependant on this and this again is the probable cause for having no change in the f1 scores.