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. The Labels being:

- O, not part of an entity
- PER, a person's name
- LOC, a locations name
- ORG, an organisations name
- MISC, a name not covered by the above labels

To build the perceptron various $\phi_n(x, y)$ have been defined, these take the words and labels of a sentence and extract features from that sentence which may be useful in predicting labels for un-seen words. ϕ_1 is based on current words and their corresponding label, this ϕ works to find the probability of a label for individual words. ϕ_2 is based on the previous label and current label. ϕ_2 focuses more on language structure and how previously seen terms influence current terms, is it more probable that a LOC tag follows a PER or ORG tag? ϕ_2 will solve this.

Furthermore; the optional bonus of two more feature types has been implemented. ϕ_3 works in a similar way to ϕ_2 although ϕ_3 is a trigram and stores information about the previous two tag labels for each current label. This feature type was chosen to further expand on how n-grams can be used to process natural language. ϕ_4 is a bigram based on the previous word and its label and the current word and its label. ϕ_4 should perform better in recognising entities made of more than one word such as New York. This entity would go unrecognised by other ϕ which do not store information on prior words and labels.

Code Implementation Explanation

Execution

The file is executed in the terminal using, *python3 lab3.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.

Feature Extraction

The *FeatureExtraction* class contains methods for extracting the features from a text corpus for each feature type. A dictionary is returned from each method containing the count of all features found. For the *currentWordCurrentLabel* method a threshold is used, this threshold removes any features in the dictionary that have a count of less than the threshold. By using a threshold of 3 reduces the feature space from over 3000 terms to just over 600 terms, this reduces strain on computer memory and increases computation speed. However; by removing terms from the feature dictionary will decrease the accuracy of predictions as in shown table 1.

Perceptron / Training

The perceptron used in the assignment is a Structured Averaged Perceptron, unlike the Perceptron implemented in assignment, this perceptron is capable of classifying over millions of possible choices. The choices being the sequence of the tags. The perceptron's task is to build the weights vector, so that each weight accurately represents the strength of the relationship for each feature. "BOB_LOC" would most likely have a negative weighting compared to "LONDON_LOC" which would have a highly positive weighting as the relationship between "LONDON" and the "LOC" tag is very strong. The weights vector is trained over multiple iterations, where for each iteration the training dataset order is randomised, randomisation stops the weights becoming generalised to a fixed order of sentences from the corpus. Removing randomisation reduces prediction accuracy by 12% to 20%.

The perceptron is implemented via two functions called *train* and *predict*. At the start of each training iteration *train* is called and each sentence in the data is iterated through. The *predict* function is passed the sentence and every possible tag combination for that sentence is attempted with the most probable sequence returned as the predicted sequence. If the tag sequence prediction is not equal to the real tag sequence for that sentence, 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 updated weights are returned and added the sum of weights which is then averaged after all iterations.

Prediction
The weights trained by the perceptron are now used, by the same *predict* method as used during training, to predict the tag sequence for the unseen training dataset. The tag predictions for each sentence are stored in *y_predicted* and the real tags for each sentence are stored in *y_true*. The sklearn f1 micro score is then applied to the two arrays to determine the accuracy of the tag predictions, f1 micro is used to evaluate the predictions as the dataset is imbalanced. Interestingly if the label array used during sequence prediction has the order *['PER', 'LOC', 'ORG', 'MISC', 'O']* returns the highest accuracy of predictions, this is again due to the imbalance of the dataset.

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

Model	Threshold	F1 Score
Phi_1	1	0.880
	3	0.803
Phi_1/2	1	0.896
	3	0.777
Phi_1/2/3/4	1	0.898
	3	0.821

Table 1: Comparison of the three Phi models and their corresponding F1 scores for a threshold of 1 and 3. Phi_1/2/3/4 achieves the highest accuracy of 0.898 when a threshold of 1 is used. All tests used 10 iterations and the 'O' label was included in model evaluation.

As shown in Table 1, over both thresholds, Phi_1/2/3/4 performs best. This model includes the optional Phis which focused on retaining prior sentence structure and therefore the more information extracted from sentences the greater the accuracy in predicting named entities. However; the addition of Phi_1/2/3/4 comes at a computational cost. The time to train and test Phi_1/2/3/4 is more than the time to train and test both the Phi_1 and Phi_1/2 models combined. This noticeable increase in time gives at best a 5.6% improvement in accuracy over the worst performing model. If accuracy of named entity prediction was critical to an application, then Phi_1/2/3/4 would be a viable but for most applications users demand both a combination of accuracy and speed something which is provided by the Phi_1 and Phi_1/2 models.

As mention above, the threshold used to determine whether terms remain in feature dictionaries vastly effects the quality of the tag prediction. The predictions produced by all models are improved when a threshold of 1 is used compared to a threshold of 3. This is due to more of the feature space being available when finding the most probable tag sequence when there is a threshold of 1. Phi_1/2 is most affected by threshold due to many terms in the feature space occurring rarely and so much of the feature space is removed. This then has a knock-on effect as some of the possible previousTag_currentTag combinations no longer exist, meaning fewer tag combinations can chain to form a tag sequence.

Table 2 contains the most positive weights obtained for each label for a given model. These results differ little between the models with the only differences occurring in the 'O' entity label section. This is because the returned results have very strong relationships between the word and corresponding tag and therefore have a highly positive weight. For example, "PETER_PER" will always have a very positive weight because it is highly probable that the word "PETER" is about a person. All of the returned results have been contributed by the Phi_1 model, Phi_1 retains the least feature information from sentences and so there is less chance of variations altering the probability of a Phi_1 feature, this a high magnitude of weight for a Phi_1 feature.

Model	Most Positive 'PER'	Most Positive 'LOC'	Most Positive 'ORG'	Most Positive 'MISC'	Most Positive 'O'
Phi_1	Peter_PER', 'Younis_PER', 'Mark_PER', 'R._PER', 'Paul_PER', ' Ahmed_PER', 'Adrian_PER', 'Warner_PER', 'Martin_PER', ' Salim_PER'	'LONDON_LOC', 'England_LOC', 'PARIS_LOC', 'AMSTERDAM_LOC', 'BONN_LOC', 'WASHINGTON_LOC', 'MOSCOW_LOC', 'BRUSSELS_LOC', 'COLOMBO_LOC', 'TOKYO_LOC'	'Newsroom_ORG', 'TEXAS_ORG', ' St_ORG', 'OAKLAND_ORG', 'CINCINNATI_ORG', 'Oakland_ORG', 'Philadelphia_ORG', 'OB_ORG', 'Milwaukee_ORG', 'Cincinnati_ORG'	'C$_MISC', 'DIVISION_MISC', 'League_MISC', 'LEAGUE_MISC', 'EASTERN_MISC', 'CENTRAL_MISC', 'English_MISC', 'Dutch_MISC', 'Baseball_MISC', 'AMERICAN_MISC'	'._O', ':_O', ',_O', '1_O', '1996-08-28_O', '0_O', '2_O', '1996-08-22_O', '1996-08-29_O', '1996-08-27_O'
Phi_1/2	'Peter_PER', 'Younis_PER', 'Mark_PER', 'R._PER', 'Paul_PER', 'Ahmed_PER', 'Adrian_PER', 'Warner_PER', 'Martin_PER', ' Salim_PER'	'LONDON_LOC', 'England_LOC', 'PARIS_LOC', 'AMSTERDAM_LOC', 'BONN_LOC', 'WASHINGTON_LOC', 'MOSCOW_LOC', 'BRUSSELS_LOC', 'COLOMBO_LOC', 'TOKYO_LOC'	'Newsroom_ORG', 'TEXAS_ORG', 'St_ORG', 'OAKLAND_ORG', 'CINCINNATI_ORG', 'Oakland_ORG', 'Philadelphia_ORG', 'OB_ORG', 'Milwaukee_ORG', 'Cincinnati_ORG'	'C$_MISC', 'DIVISION_MISC', 'League_MISC', 'LEAGUE_MISC', 'EASTERN_MISC', 'CENTRAL_MISC', 'English_MISC', 'Dutch_MISC', 'Baseball_MISC', 'AMERICAN_MISC'	':_O', ',_O', 'AT_O', 'of_O', 'Results_O', 'Attendance_O', 'Women_O', 'Division_O', 'Sunday_O', '2_O'
Phi_1/2/3/4	'Peter_PER', 'Younis_PER', 'Mark_PER', 'R._PER', 'Paul_PER', 'Ahmed_PER', 'Adrian_PER', 'Warner_PER', 'Martin_PER', 'Salim_PER'	'LONDON_LOC', 'England_LOC', 'PARIS_LOC', 'AMSTERDAM_LOC', 'BONN_LOC', 'WASHINGTON_LOC', 'MOSCOW_LOC', 'BRUSSELS_LOC', 'COLOMBO_LOC', 'TOKYO_LOC'	'Newsroom_ORG', 'TEXAS_ORG', 'St_ORG', 'OAKLAND_ORG', 'CINCINNATI_ORG', 'Oakland_ORG', 'Philadelphia_ORG', 'OB_ORG', 'Milwaukee_ORG', 'Cincinnati_ORG'	'C$_MISC', 'DIVISION_MISC', 'League_MISC', 'LEAGUE_MISC', 'EASTERN_MISC', 'CENTRAL_MISC', 'English_MISC', 'Dutch_MISC', 'Baseball_MISC', 'AMERICAN_MISC'	':_O', ',_O', 'AT_O', 'of_O', 'Results_O', 'Attendance_O', 'Women_O', 'Division_O', 'Sunday_O', '2_O'

Table 2: Comparison of the top 10 terms with the most positive weight for each of the models used.