Chunking
Assignment 2

Anshul Tomar 170070007
Anwesh Mohanty 170070009
Parth Shettiwar 170070021

September 2020

Contents

1 Maximum Entropy Markov Models

1.1 Dataset preparation

For finding the feature vectors for each word, we decided to use various properties of the word combined with its word embeddings. After careful selection of all features, we decided to choose the following word representation of feature length(fl)= 110

Feature	Type	Use	Length
Word Embedding	Continous, [-1,1]	This was used mainly to use the semantics of sentences	100
POS Tags	One Hot, length 44, {0,1}	The one hot encoding of POS tag of the word	44
Number present	Binary, {0,1}	Checked whether the word contains a number	1
First Upper Caps	Binary, {0,1}	Checked whether first letter of word is Uppercase	1
Word Length	Binary, {0,1}	If length of word is less <= 3, then set to 1	1
Punctuation	Binary, {0,1}	Set to 1 if word contains punctuation.	1
Hyphen	Binary, {0,1}	Set to 1, if word contains hyphen	1
Suffix	One Hot, length 4, {0,1}	Common suffixes corresponding to Adjective, Noun, Adverb, Verb are checked for in word and one hot vector formed	4
Possession	Binary, {0,1}	Checked for {'} character in word	1
		Total Length	154

As mentioned in table, we created a list of suffixes for each Noun, Verb, Adverb and Adjective tags, which are prevalent.Specifically 16 for Noun, 10 for Verb, 3 for Adverb and 13 for Adjective. The whole dataset was prepared manually(forfinding features of each word). For word embeddings we used the gensim library Word2Vec and trained our corpus on it.Finally a window of 2 was taken, where we append the above made word features of adjacent words, finally giving 154*2 =308 features for each word. For end cases(end words of sentence), such added features were just set to 0 and tags were set to 'START'.

1.2 Implementation

The MEMM model was implemented using the inbuilt model in NLTK library called as MaxentClassifier. The model is trained for 10 epochs as each epoch takes a large amount of time. The sequence is decoded using viterbi's backpropogation algorithm.

1.3 Results and error analysis

- We first show the confusion matrix from which all accuracies can be easily obtained. "O" tags were kept inside the dataset and not removed, as was instructed.

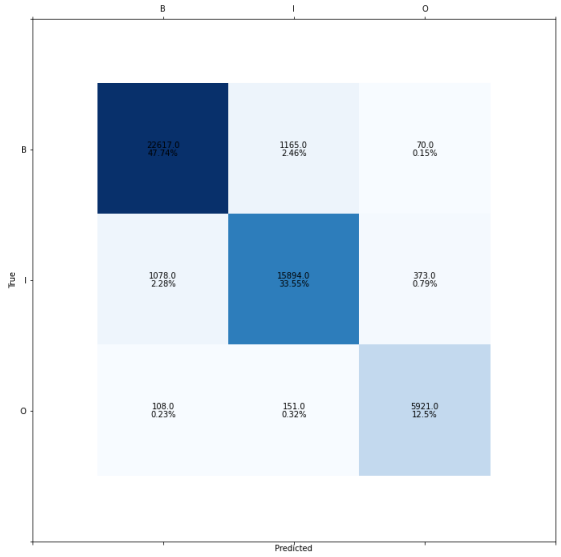

Figure 1: Confusion Matrix for MEMM

- From this matrix, we can easily calculate the various accuracies required. Based on the confusion matrix, the final results obtained are-

Tag	Precision	Recall	F1 Score
B	0.950	0.948	0.949
I	0.923	0.916	0.919
O	0.930	0.958	0.944

The overall accuracy comes out to be 93.78%.

	Precision	Recall	F1 Score
Overall	0.9376	0.9379	0.9378

- Finally the overall accuracy can also be similarly calculated. In this we dont account for the accuracy for "O" and Padtags, as they are not required. Hence the overall accuracy is .

Following points for error analysis can be noted

- Both B and I tags enjoy a high accuracy of greater than 90%. Also F1 score, precision and recall for B are in general higher than I tag.
- The overall accuracy is relatively closer to the B tag, as there are relatively more B tags in the dataset.
- One of the reasons for relatively lesser scores for I tag, might be due to less number of instances (as compared to B tag). Due to presence of O tag also, both recall and precision of I tag are affected by small amount. Its also affecting B tag, but as said, due to higher number of training data for B tag, their effect is relatively lesser.
- The B and I tags are acting as each others highest False positive and False negatives due to obvious reasons.

2 Conditional Random Fields

2.1 Dataset Preparation

For finding the feature vectors for each word, we decided to use various properties of the word combined with word embeddings of the word. After careful selection of all features, we decided to choose the following word representation of feature length(f_l) = 110

Feature	Type	Use	Length
Word Embedding	Continous, [-1,1]	This was used mainly to use the semantics of sentences	100
POS Tags	One Hot, length 44, {0,1}	The one hot encoding of POS tag of the word	44
Number present	Binary, {0,1}	Checked whether the word contains a number	1
First Upper Caps	Binary, {0,1}	Checked whether first letter of word is Uppercase	1
Word Length	Binary, {0,1}	If length of word is less <= 3, then set to 1	1
Punctuation	Binary, {0,1}	Set to 1 if word contains punctuation.	1
Hyphen	Binary, {0,1}	Set to 1, if word contains hyphen	1
Suffix	One Hot, length 4, {0,1}	Common suffixes corresponding to Adjective, Noun, Adverb, Verb are checked for in word and one hot vector formed	4
Possession	Binary, {0,1}	Checked for {'} character in word	1
		Total Length	154

As mentioned in table, we created a list of suffixes for each Noun, Verb, Adverb and Adjective tags, which are prevalent. Specifically 16 for Noun, 10 for Verb, 3 for Adverb and 13 for Adjective. The whole dataset was prepared manually(for finding features of each word). For word embeddings we used the gensim library Word2Vec and trained our corpus on it. Finally a window of 5 was taken, where we append the above made word features of adjacent words, finally giving 154*5 = 770 features for each word. For end cases(end words of sentence), such added features were just set to 0.

2.2 Implementation

The CRF was implemented using the inbuilt keras layer "CRF" which is created in the repo "keras-contrib". We have used the "join mode" of training, which is the required mode for maximimsing the CRF likelihood function. The training is done using negalitve log-likelood as the loss function. The label encoding is used for target labels, with 0 for "B" tag, 1 for "I" tag and 2 for "O" tag. A sentence is feeded into the model with the 770 length feature set described in previous section. For uniform dimension input, add sentences were padded with appropriate zeros to make the final input dimension to be (Batch Size, 78, 770) where 78 is the max length of sentence in the corpus. For this padding, the output label is kept to be "3" while making the dataset. A split of about 85:15 is made for training:validation sets. We train our model for 15 epochs(as after that the performance of model saturates). Finally for prediction, we have used the viterbi decoding.

2.3 Resuls and Error Analysis

- We first show the confusion matrix from which all accuracies can be easily obtained. As mentioned earlier, we have used the padding, hence the last row is seen. "O" tags were kept inside the dataset and not removed, as was instructed. Hence the final size of matrix is 4*4.

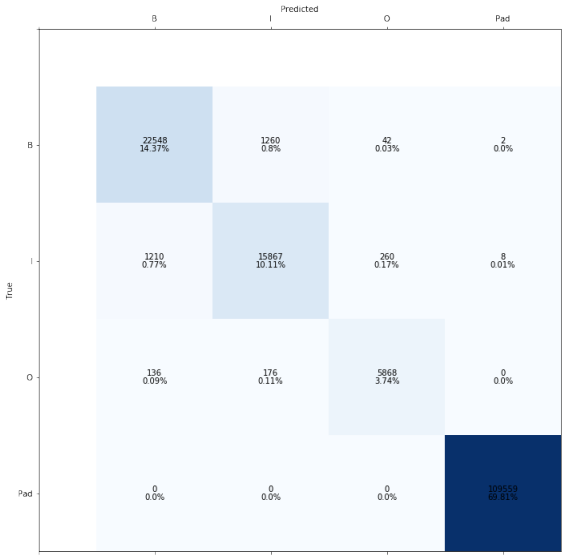

Figure 2: Confusion Matrix for CRF

- From this matrix, we can easily calculate the various accuracies required.

Tag	Precision	Recall	F1 Score
B	0.944	0.945	0.9445
I	0.917	0.915	0.916
O	0.951	0.949	0.950

- Finally the overall accuracy can also be similarly calculated. In this we dont account for the factor of Pad tags, as they are not required. We have used the weighted formula for calculating overall accuracy , precision and recall.

	Precision	Recall	Accuracy
Overall	0.9352	0.9347	0.9348

Following points for error analysis can be noted

- Both B and I tags enjoy a high accuracy of greater than 90%. Also F1 score, precision and recall for B are in general higher than I tag.
- The overall accuracy is relatively closer to the B tag, as there are relatively more B tags in the dataset.
- One of the reasons for relatively lesser scores for I tag, might be due to less number of instances (as compared to B tag). Due to presence of O tag also, both recall and precision of I tag are affected by small amount. Its also affecting B tag, but as said, due to higher number of training data for B tag, their effect is relatively lesser.
- The B and I tags are acting as each others highest False positive and False negatives due to obvious reasons.
- We analyse the effect of Pad tag. It was predicted, the introduction of Pad tag might lower the performance of model. However as seen from confusion matrix, only 10 instances (as false positives for Pad tag and 0 false negatives) are seen. Hence we conclude, the Pad tag didn't affect much the overall performance of model. However, it might have affected the weights during training, which indirectly might have affected the B and I tags accuracy, but this cant be told from looking at confusion matrix. In any case, all the stats for Overall shown are by including only B, I and O tags. Also O tag doesnt affect the overall accuracy stats, as their number of instances in corpus is very less. So the overall stats convey an accurate picture for analysis of B and I tags.

2.4 Learning

Due to the availability of inbuilt keras CRF layer, the implementation of the algorithm didnt take much time and we could spend more time on feature engineering. The final idea of using morph features, embeddings and POS tag was fruitful as it yielded high accuracies. As was done in SVM, we understood feature engineering is always crucial for any NLP task and sufficient time must be devoted on it. Including features from adjacent words helps a lot as they give context of the word in picture. We also explored the conll2000 dataset and got to know its structure.

3 Bidirectional Long-Short Term Memory (BiLSTM)

3.1 Pre-processing Data

To develop the feature vectors of each word in the CoNLL2000 dataset provided to us, we use various properties of the words along with their embeddings (found using Word2Vec after training on the corpus). The features used are the same as mentioned in the Section2.1 (to perform a good comparison between both models, also the accuracy obtained with the chosen features were good enough so we did not have to search for other better features).

Feature	Type	Use	Length
Word Embedding	Continous, [-1,1]	This was used mainly to use the semantics of sentences	100
POS Tags	One Hot, length 44, {0,1}	The one hot encoding of POS tag of the word	44
Number present	Binary, {0,1}	Checked whether the word contains a number	1
First Upper Caps	Binary, {0,1}	Checked whether first letter of word is Uppercase	1
Word Length	Binary, {0,1}	If length of word is less <= 3, then set to 1	1
Punctuation	Binary, {0,1}	Set to 1 if word contains punctuation.	1
Hyphen	Binary, {0,1}	Set to 1, if word contains hyphen	1
Suffix	One Hot, length 4, {0,1}	Common suffixes corresponding to Adjective, Noun, Adverb, Verb are checked for in word and one hot vector formed	4
Possession	Binary, {0,1}	Checked for {'} character in word	1
		Total Length	154

The total feature length for any word hence is 154. For the BiLSTM we have used a window of 3 (appended the features of 2 adjacent words to the current word) to give a total of 154*3 = 462 features for a single word. For words at the ends of sentences we simply append zeros at the end.

3.2 Model Architecture

Layers in the proposed BiLSTM network:

- **BiLSTM Layer**: LSTM layer with bidirectional modifier; the modifier inputs the next values in the sequence along with the previous values to the LSTM. It takes in word embeddings as input and outputs hidden states with dimensionality hidden_dim (set by us).
- **Dense Layer**: This layers maps from the hidden state space to the tag space. We process this output through a softmax layer to pick up the appropriate POS tag.

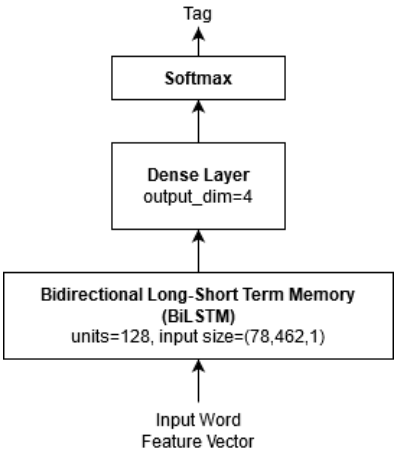

Figure 3: Architecture of our BiLSTM network

Model Flow: Let our input sentence be $w_1, \ldots, w_M$, where $w_i \in V$, our vocabulary of words. Also, let T be our chunk set, and y_i the chunk of word w_i. Let our prediction of the chunk of word w_i be denoted by y_i. The corresponding output for our input sentence will be a sequence $y_1, ..., y_M$, where $y_i \in T$.
To get our prediction, we pass the input sentence through our model. Let the hidden state at timestep i be denoted as h_i. Our prediction rule for each y_i is:

$$y_i = argmax_j(Softmax(Ah_i + b))_j \tag{1}$$

i.e we take the softmax of the affine map of the hidden state, and the predicted chunk is the chunk that has the maximum value in this vector. (Also note that this indicates that the dimensionality of the target space of A is $|T|$.)

Implementation: The entire model has been built using the in-built LSTM class present in Keras. We have used the Adam optimizer and the loss metric used is the categorical crossentropy loss since it is a classification task. For the output, 4 labels are used: 0 for 'B', 1 for 'I', 2 for 'O' and 3 for 'Pad'. Each sentence in the input is padded to the maximum length (78) of a sentence present in the dataset by padding them at the end. The 'PAD' tag is given to all such padded elements. Each word in a sentence has 154 features and we use a window size of 3 which makes the entire feature length of a single word to be 462. Hence for each sentence the input size is (78,462) and hence the input shape in the BiLSTM architecture (Fig3). We use a 85:15 split for training and validation. The model is run for 10 epochs and the final output is demonstrated in the form of a confusion matrix.

3.3 Results

- First we show the final confusion matrix obtained after running the model for 10 epochs. Since we have 4 labels ('B', 'I', 'O' and 'PAD') the size of the confusion matrix is 4×4. We don't report the results on the 'PAD' tag as they have been introduced by us for our convenience.
- Based on the confusion matrix results, the precision, recall and F1 score for the 'B', 'I' and 'O' tags are-

Tag	Precision	Recall	F1 Score
B	0.9692	0.9663	0.9677
I	0.9472	0.9521	0.9497
O	0.958	0.955	0.9565

- The overall accuracy comes out to be 95.96%. For the overall precision, recall and F1 score we use the weighted method i.e calculate metrics for each label, and find their average weighted by support (the number of true instances for each label).
  - Overall Precision = 0.9598
  - Overall Recall = 0.9597
  - Overall F1 score = 0.95971

3.4 Error Analysis

- All the three tags i.e 'B', 'I' and 'O' are predicted quite accurately by the model after just training for 10 epochs with greater than 95% accuracy for each tag. This makes it the best model out of the three models taken for comparison.
- The overall accuracy is closer to the 'B' tag due to more occurrences of 'B' tag. The accuracy of 'I' is slightly lower than that of 'B' which might be due to its reduced instances in the dataset compared to 'B'.
- The final accuracy of 'B' and 'I' tags are also affected slightly due to the presence of the 'O' tag. But since instances of 'O' tag are much smaller compared to other two, the impact on accuracy is not that much.
- The highest number of false positives and false negatives are due to mis-classification of 'B' as 'I' and vice-versa. And since the number of 'O' tags present are much lower compared to 'B' and 'I', the mis-classification of 'O' does not affect accuracy that much.
- Since we introduce padding into our model we also have to consider the change in accuracy introduced due to padding. But as can be seen from the confusion matrix, ALL the padding tags have been classified with 100% accuracy with no false positives or negatives. This indicates that the model is completely robust to the padding tags and is not affected by such tags.

3.5 Possible Scopes for Improvement

- Increase the number of hidden layers in the BiLSTM. For our model we have chosen the number of hidden dimensions as 128. As we increase this hidden dimension, the final accuracy will improve albeit slightly but increase nonetheless. But then we will have to take in consideration the computation time as increasing the hidden dimension will lead to increase in computation time and resources required as well.
- Increasing the total number of epochs might give us better results as the model might learn the underlying complex distribution of the dataset much better due to increased epochs.

- Other different features can be added during the feature engineering phase to enhance the performance. The window length can be increased from 3 to other higher values to take into consideration the interaction of words with their neighbouring words.
- The loss function can be modified according to recent work in the field to improve the accuracy on outliers as well as improve upon the overall accuracy in general.

3.6 Learning

The main learning in this assignment was the importance of feature learning in NLP tasks and how including/excluding certain features can affect the final accuracy drastically. We also got a flavour of working with the CoNLL2000 dataset.

4 Conclusion

The 3 methods for Chunking have been implemented with appropriate feature engineering.