POS Tagging
Assignment 1

Anshul Tomar 170070007
Anwesh Mohanty 170070009
Parth Shettiwar 170070021

September 2020

Contents

1 Hidden Markov Models

1.1 Assumptions taken and modifications done

The HMM algorithm was taught in class. We have implemented the same algorithm as was described. The transition and emission probabilities were found in the exact same way as was taught in the class except for these modifications.

- The sentence always starts with a 'START' tag and always ends with a '.' tag.
- Applied add-α smoothing in which all the OOV words were initialized with $\frac{1}{V}$ where V is the vocabulary size.
- Also boosted all the probability values by a boostingFactor after every boostingEpoch as the probaility values rapidly decline as we go along a long sentence.

1.2 Results

The per class accuracy on validation set after 5 fold cv are given in table 1. The confusion matrix is given in figure 1.

Tag	Accuracy(in %)
NOUN	88.18
PRT	89.91,
PRON	98.40
X	63.34
.	100 [1]
ADJ	90.12
VERB	94.62
NUM	90.34
ADV	89.62
ADP	96.31
DET	98.66
CONJ	98.82
START	100
All tags	94.28

Fold Number	Accuracy(in %)
1	93.63
2	94.16
3	94.47
4	94.72
5	94.43
Average	94.28

Table 1: Tag wise accuracies and per fold accuracies for the task of POS tagging using HMM

1.3 Error Analysis

The HMM algorithm yields a very high accuracy of 94.28%. For all tags except X tag we have 80+ % accuracy in each atleast. Further for each tag false negatives are very less. Also for X tag, the accuracy is highest compared to other models. The 5 fold accuracy is also very stable.

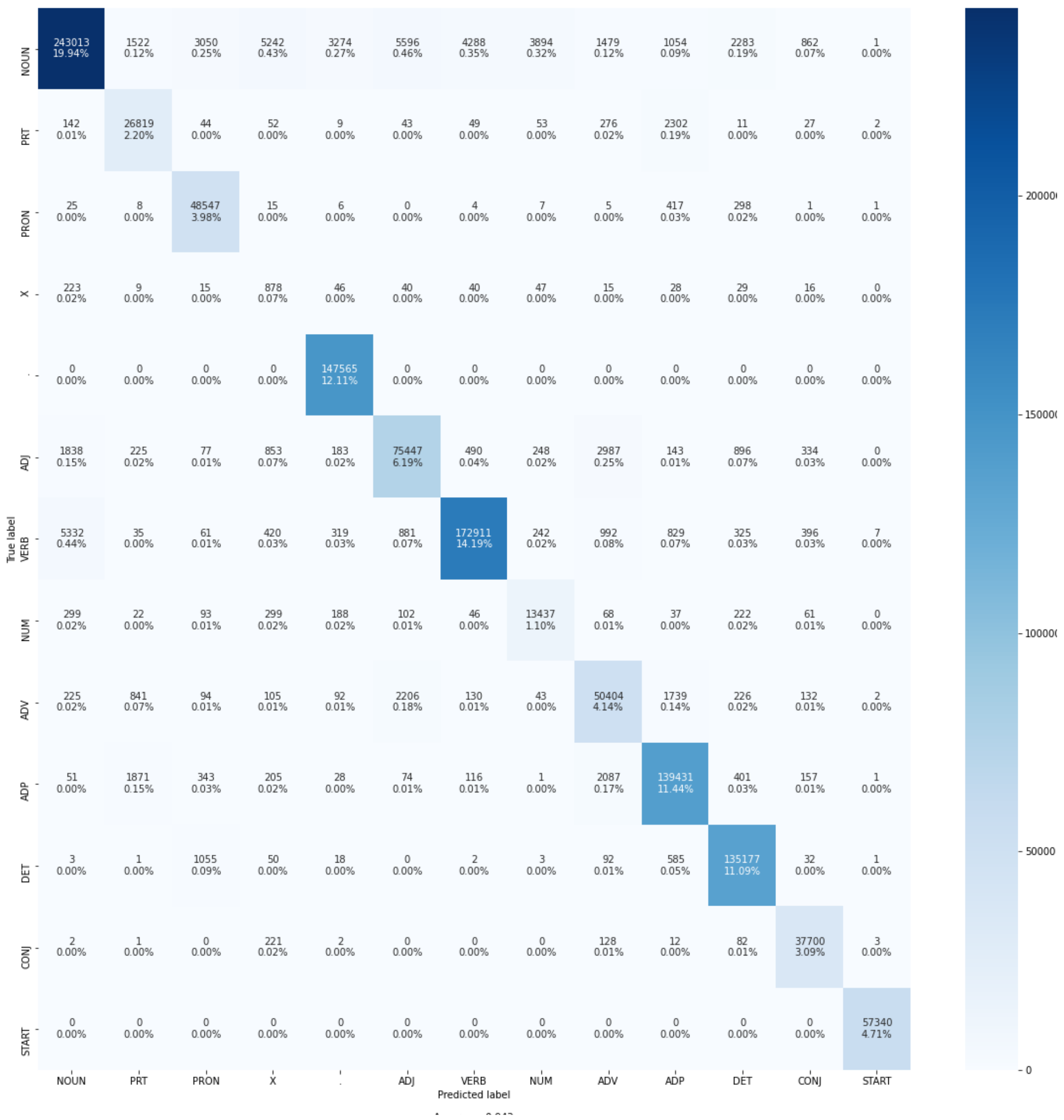
1.4 Strengths and Weaknesses

Following are the strengths of the HMM algo

- The algorothm gives a very high accuracy of 94.28 %. Further the accuracy is more or less good for individual POS tags.
- The time and space complexity for HMM algo are very less. Training time is hence very less compared to other algos.
- Not much feature engineering required and no hyperparamters as such, which is unlike other algos.

Following are the weaknesses of HMM algo

- The HMM is a ML algo hence, it has less power/potential to learn the underlying distribution of very complex datasets. Hence we have Bi-LSTMs nowadays which are Deep neaural networks which are giving State of Art accuracy on various complex datasets.
- The HMM algo needs to see each instance at least once, and also not robust to outliers.



	NOUN	PRT	PRON	X	.	ADJ	VERB	NUM	ADV	ADP	DET	CONJ	START
NOUN	243013 19.94%	1522 0.12%	3050 0.25%	5242 0.43%	3274 0.27%	5596 0.46%	4288 0.35%	3894 0.32%	1479 0.12%	1054 0.09%	2283 0.19%	862 0.07%	1 0.00%
PRT	142 0.01%	26819 2.20%	44 0.00%	52 0.00%	9 0.00%	43 0.00%	49 0.00%	53 0.00%	276 0.02%	2302 0.19%	11 0.00%	27 0.00%	2 0.00%
PRON	25 0.00%	8 0.00%	48547 3.98%	15 0.00%	6 0.00%	0 0.00%	4 0.00%	7 0.00%	5 0.00%	417 0.03%	298 0.02%	1 0.00%	1 0.00%
X	223 0.02%	9 0.00%	15 0.00%	878 0.07%	46 0.00%	40 0.00%	40 0.00%	47 0.00%	15 0.00%	28 0.00%	29 0.00%	16 0.00%	0 0.00%
.	0 0.00%	0 0.00%	0 0.00%	0 0.00%	147565 12.11%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%
ADJ	1838 0.15%	225 0.02%	77 0.01%	853 0.07%	183 0.02%	75447 6.19%	490 0.04%	248 0.02%	2987 0.25%	143 0.01%	896 0.07%	334 0.03%	0 0.00%
VERB	5332 0.44%	35 0.00%	61 0.01%	420 0.03%	319 0.03%	881 0.07%	172911 14.19%	242 0.02%	992 0.08%	829 0.07%	325 0.03%	396 0.03%	7 0.00%
NUM	299 0.02%	22 0.00%	93 0.01%	299 0.02%	188 0.02%	102 0.01%	46 0.00%	13437 1.10%	68 0.01%	37 0.00%	222 0.02%	61 0.01%	0 0.00%
ADV	225 0.02%	841 0.07%	94 0.01%	105 0.01%	92 0.01%	2206 0.18%	130 0.01%	43 0.00%	50404 4.14%	1739 0.14%	226 0.02%	132 0.01%	2 0.00%
ADP	51 0.00%	1871 0.15%	343 0.03%	205 0.02%	28 0.00%	74 0.01%	116 0.01%	1 0.00%	2087 0.17%	139431 11.44%	401 0.03%	157 0.01%	1 0.00%
DET	3 0.00%	1 0.00%	1055 0.09%	50 0.00%	18 0.00%	0 0.00%	2 0.00%	3 0.00%	92 0.01%	585 0.05%	135177 11.09%	32 0.00%	1 0.00%
CONJ	2 0.00%	1 0.00%	0 0.00%	221 0.02%	2 0.00%	0 0.00%	0 0.00%	0 0.00%	128 0.01%	12 0.00%	82 0.01%	37700 3.09%	3 0.00%
START	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	57340 4.71%

Figure 1: Confusion matrix for HMM POS tagging

2 Support Vector Machines

2.1 Approach

The Supper Vector Machines learn classifiers so as to classify the Tag of a word in a sentence. The loss function optimization of SVM has been done using various approaches in past. The classifiers are learnt such that they have a maximum width margin learnt so as to easily classify the samples with some outliers. For POS tagging, it is very crucial how the features for each word are prepared so as to learn appropriate classifiers. So we divided the whole Approach into 2 parts: Dataset Preparation and Implementation.

2.1.1 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	Intuition of Use	Length
Word Embedding	Continous, [-1,1]	This was used mainly to use the semantics of sentences	100
Number present	Binary, {0,1}	Checked whether the word contains a number, helpful for identifying ”NUM” Tag	1
First Upper Caps	Binary, {0,1}	Checked whether first letter of word is Uppercase, helpful usually in identifying Nouns	1
Word Length	Binary, {0,1}	If length of word is less <= 3, then set to 1, helpful in Determinars and Adposition	1
Punctuation	Binary, {0,1}	Set to 1 if word contains punctuation, help in ”.” tag.	1
Hyphen	Binary, {0,1}	Set to 1, if word contains hyphen, help in usually adjective	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, helps to identify PRT tags	1
		Total Length	110

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 3 was taken, where we append the above made word features of adjacent words, finally giving 110*3 = 330 features for each word. For end cases(first and last word of sentence), such added features were just set to 0.

2.1.2 Implementation

The SVM was implemented from the paper ”Large-scale Multiclass Support Vector Machine Training via Euclidean Projection onto the Simplex”. The current state of art techqniue for solving SVMs are using dual decomposition methods. Since the training time would be very huge considering our corpus, it was of utmost necessity to implement the fastest algo too. The above paper proposes to solve the Dual variables update in exactly $O(k)$ time where k is the number of classes. The rest of algos, had higher time complexity. Specifically the parameter to update the dual variables was solved using Algorithm 3(Bisection for Projection) as proposed in paper. The overall Dual Decomposition was done using Algorithm 1.

2.1.3 Error Analysis

- Ablation Study: The correct feature set for SVM was a bit tricky task. Hence we performed mainly 3 experiments to finally experimentally show, our 110 feature set for each word yields the highest accuracy using SVMs. Firstly, we took only 10 features for each word and removed the 100 length word embedding. Second experiment included only word embeddings for each word, and finally a combination of both.

Experiment	Accuracy
Properties of word	46-48%
Word Embeddings	82-84%
Word Embeddings + Properties of word	88-90%

  We observe that combination of both is desirable and achievse highest accuracy.

- With combination of both word embeddings and properties of word used, we finally went ahead with calculating the Per-Pos accuracy and 5 Fold accuracy for our setup. Except the ”X” tag which is usually identified by no meaningful property, and PRT(possessor) tag, we achieve a 70+% accuracy on each of the tags.

TAG	Accuracy(in %)
VERB	87.20
NOUN	83.97
PRON	96.64
ADJ	79.00
ADV	72.48
DET	98.76
ADP	96.44
PRT	39.39
NUM	81.60
CONJ	98.28
X	18.75
.	99.99

A further 5 fold accuracy is calculated and we observe with minor changes in first decimal, we always get a stable around 88-89% accuracy always. The final average is reported as 88.6%.

Fold Number	Accuracy(in %)
1	88.55
2	89.73
3	88.78
4	87.83
5	88.38
Average	88.66

We finally plot the confusion matrix for the various tags.

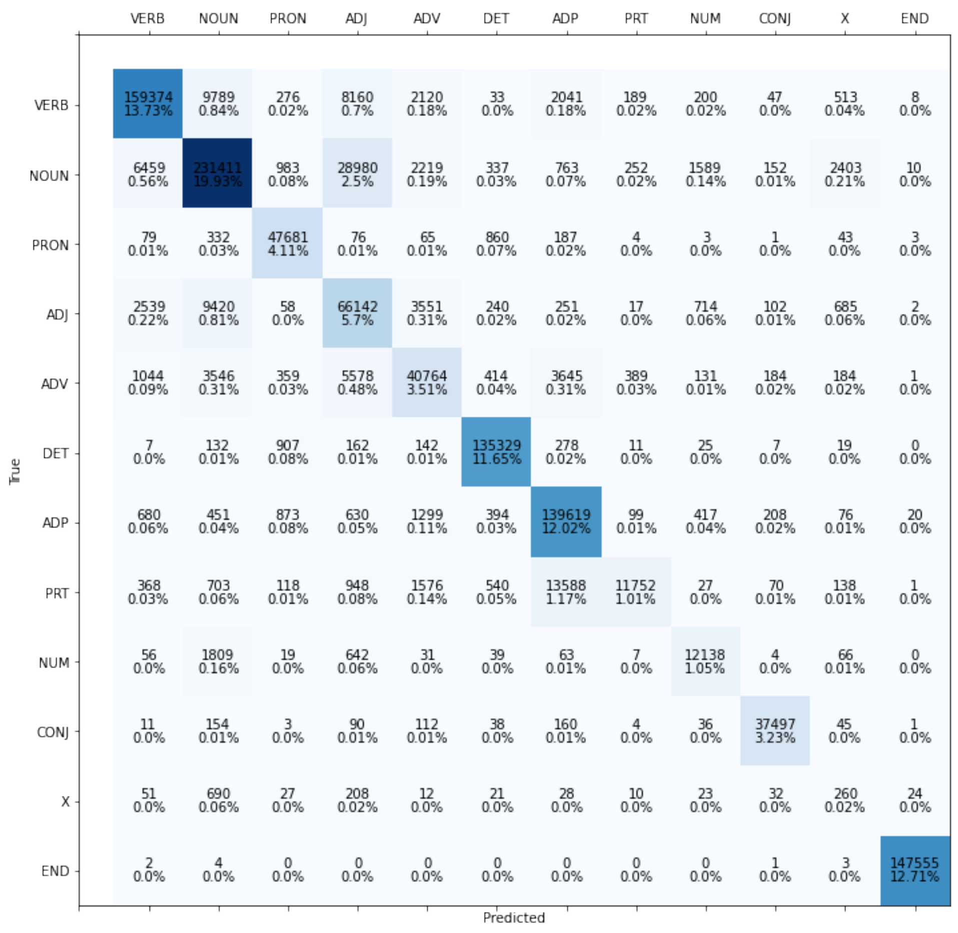


Figure 2: Confusion Matrix for SVM POS tagging, on y-axis we have True labels and on x-axis we have predicted labels. The percent written below are just the percent of words classified as that tag and with that true label of total corpus.

Following points can be noted

- Verbs and nouns contribute highest percentage of corpus relative to other labels. Verbs if wrongly classified, are mainly nouns. Similarly, in Nouns, the highest incorrect label is Adjective.
- For rest, the words are already less in corpus, so sometimes their accuracy is low, since much trainig data was not there, for example X tag has very low accuracy.
- With Case wise analysis of where accuracy is dropping, a more detailed featre engineering can be adopted to solve these corner cases.

2.2 Strength and Weaknesses

The SVM assignment entailed lot of elements at different stack of NLP. We used semantics and morphology.
The following are the Weaknesses of SVM implementation:

- The training time is huge(takes almost 2-2.5 hours to converge) even after using the state of art $O(k)$ time algorithm. As the lenth of vector increases, the time also increases proportionally.
- The feature vector is crucial in SVM algo. A lot of hyperparamters are there and need to be varied to know what suits the best.
- The max accuracy achieved even after deliberate and careful feature vector engineering is max around 88%(can be improved though). Further the accuracy is not same on all tags. For some tags like ”X”, the accuracy was mere 18.75%.

The following are the Strengths of SVM implementation:

- Unlike neural networks, the number of parameters/weights used in SVM algo is very less(as its a machine learning algorithm). Hence the space complexity required is less.
- With enough data and good feature engineering, it can give very good accuracies. This was observed by some tags like ”PRON”, ”ADP” and ”CONJ”.
- The SVM algo is in general robust to outliers.

2.3 Learning

SVM implementation took the highest amount of time compared to other algos. This was mainly because a lot of feature engineering was required and to identify the correct features required. This was complicated by identifying the most efficient algorithm to lessen the training time. The algorithm for optimization was particularly new and took some time to understand the underlying maths behind it. A comprehensive ablation study was performed on what features to use, the window size to be used and various regularization parameters of the algorithm. A three way merge of different types of feature type, namely: One hot, Binary, and Continous between [-1,1] yield fruitful. The final comprehensive error analysis gave a great level of satisfaction on the implementation and made us think ways to improve accuracies for each tag aided by confusion matrix plot. It was a holistic learning where from scratch all things were carefully thought of and culminated in a fair accuracy. With more time spent on feature engineering, we could have definitely improved upon the current highest accuracy by 1 or 2 points.

3 Bidirectional Long-Short Term Memory (BiLSTM)

3.1 Introduction

A recurrent neural network is a network that maintains some kind of state. For example, its output could be used as part of the next input, so that information can propagate along as the network passes over the sequence. In the case of an LSTM, for each element in the sequence, there is a corresponding hidden state h_t, which in principle can contain information from arbitrary points earlier in the sequence. We can use the hidden state to predict part-of-speech tags. In this section we build a BiLSTM in PyTorch as they have been shown to be very effective for tagging sequential data.

3.2 Pre-processing Data

We use the Brown corpus included in the nltk package as our dataset and the set of tags is chosen as the ”universal tagset”. Since we cannot directly feed words or tags (strings or characters) to a model in Pytorch, we need to convert them into numbers. We iterate through the dataset and assign indices to unique words and tags, and basically convert them into a list and index them in a dictionary. These dictionaries are the word and tag vocabulary. Now each word and the corresponding tag in a sentence can be expressed as numbers which can then be fed into the BiLSTM.

3.3 Model Architecture

Layers in the proposed BiLSTM network:

- **Embedding Layer**: Computes a word vector model for our words. Embeddings are stored as a $|V| \times D$ matrix, where D is the dimensionality of the embeddings, such that the word assigned index i has its embedding stored in the i^{th} row of the matrix. This layer will take the take in a sequence of words and convert them into the word embedding space.
- **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 log-softmax layer to pick up the appropriate POS tag.

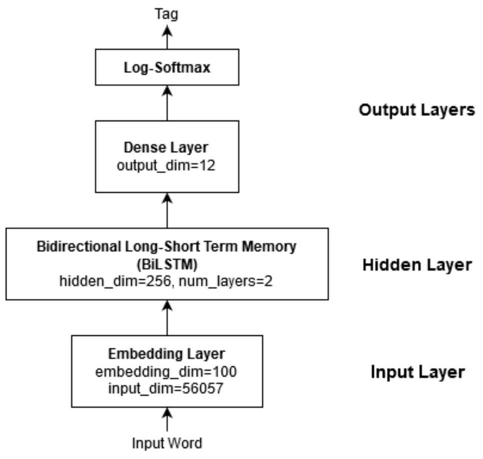


Figure 3: Architecture of our BiLSTM network

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 tag set, and y_i the tag of word w_i. Let our prediction of the tag 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 ech y_i is:

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

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

3.4 Results and Error Analysis

The per class accuracy on validation set after 5 fold cv and accuracy for each fold are given in the below tables. The confusion matrix is given in figure 3.

TAG	Accuracy(in %)
DET	98.95
NOUN	94.18
ADJ	72.86
VERB	88.75
ADP	94.35
.	99.89
ADV	80.31
CONJ	98.65
PRT	84.24
PRON	96.74
NUM	59.97
X	1.23

Fold Number	Accuracy(in %)
1	91.32
2	91.73
3	92.05
4	92.33
5	92.08
Average	91.90

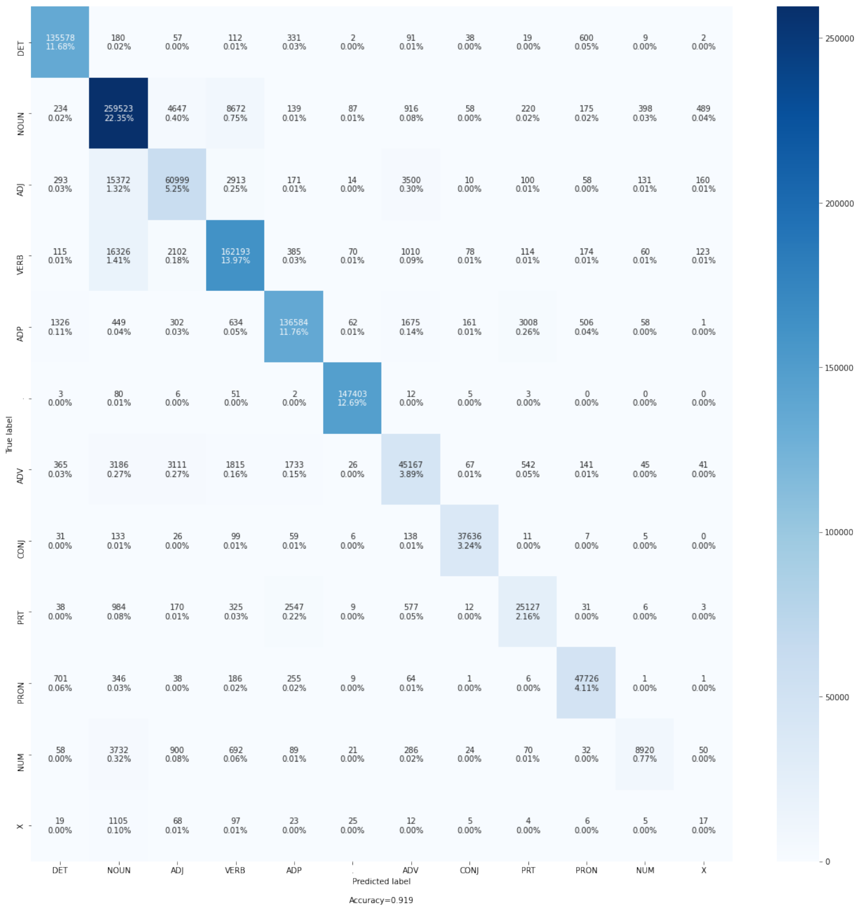


Figure 4: Confusion Matrix for BiLSTM

From the table for individual accuracy of each tag, the following observations can be made-

- The model performs very well for 'PRON', 'NOUN', 'DET', 'ADP' and 'CONJ' as evidenced by the > 94% classification accuracy .
- There is a dick in performance for adjectives ('ADJ') (almost 30% samples are classified wrongly). From the confusion matrix it can be seen that most of the adjectives if wrongly classified, are classified as nouns. Similarly, in the 11% of verbs that are wrongly classified, have been classified as nouns. Moreover, most of the misclassified nouns are either marked as verbs or adjective.

- Of the 40% numbers ('NUM') that are wrongly classified, most of them have been classified as nouns.
- The model shows very poor accuracy on the 'X' tag, which may be due to the sparsity of words belonging to the 'X' tag set.

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 100 based on the paper "Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network". As provided in their analysis, 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.
- Modifications in the embedding layer (increasing the embedding dimension) might produce better results but we were unable to pursue this due to time constraints.
- In every fold we have only run the model for 5 epochs due to the large amount of time required for each epoch. Hence if the model is allowed to run for 20-30 epochs the network might be able to train better on the dataset and provide better results; especially enhance the results for the poor accuracy on tags like "NUM", "X" and "ADJ" for example.
- 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 Strengths and Weaknesses

The strengths of the BiLSTM network are:

- Being a deep neural network model, the BiLSTM can give good results even with large datasets consisting of several words and a large tagset. Infact as the size of the tagset increases, the performance of the BiLSTM model over HMM and SVM also increases. That is why the state-of-the-art accuracy in POS tagging for several datasets are achieved by using different variation of BiLSTMs.
- Minimal manual tuning of parameters is required(compared to SVM) as the neural network learns by itself based on the loss metrics.

The weaknesses of the BiLSTM network are:

- Being a deep neural network model, the memory footprint as well as the time complexity of the network is quite high. Hence training of the model takes a lot of time as well as resources.
- Poor performance on a tag if there are very few instances of it (for example the X tag for our case).

3.7 Learning

The BiLSTM implementation of the project was different than others in the sense that none of us actually knew how BiLSTMs operated before this (we just knew that LSTMs are a type of RNN and RNNs are used for sequential data). So for this we had to read up a few papers on this to get the basic idea of how BiLSTMs can be used for POS tagging. We now have great understanding of the maths underlying Bi-LSTMs and why they are adopted in Language Learning Tasks. Due to resource and time constraints, we were not able to play around with the model parameters and epochs, which might have given us better results and further insights into the project.

4 Comparing all three algorithms for POS tagging

TAG	HMM (in %)	SVM (in %)	BiLSTM(in %)
NOUN	88.18	83.97	94.18
PRT	89.91	39.39	84.24
PRON	98.40	96.64	96.74
X	63.34	18.75	1.23
.	100	99.99	99.89
ADJ	90.12	79.00	72.86
VERB	94.62	87.20	88.75
NUM	90.34	81.60	59.97
ADV	89.62	72.48	80.31
ADP	96.31	96.44	94.35
DET	98.66	98.76	98.95
CONJ	98.82	98.28	98.65
START	100	-	-
All tags	94.28	88.66	91.90

Table 2: Comparing the accuracies obtained from all three algorithms for POS tagging

The following analysis can be made on comparison of all 3 models for POS tagging:

- HMM yields the highest accuracy of all 3 algos. Followed by Bi-LSTM and SVM. SVM has lot of feature engineering which leads to a lesser accuracy compared to other models. HMM doesnt entail any of feature vector engineering as such.
- Reportedy X tag has lowest accuracy in each algo, in that too, HMM beats other too by great margin.
- For HMM lowest fluctuations in individual POS accuracy is observed followed by SVM and Bi-LSTM.
- All three models gave a stable 5 fold accuracy, implying they are robust enough for all data.
- Except HMM, other 2 algos are very sensitive to different hyperparamters and hence give a varying accuracy on a dataset. However, SVM and Bi-LSTM are robust to outliers but HMM needs to see atleast see all possible instances at least once.
- Time and Space complexity of Bi-LSTM are highest, followed by SVM and lastly by HMM. But at same time Bi-LSTM has potential to mimic complex distributions also if proper parameters are set, which is not in hands of other 2 ML algos.

5 Conclusion

The 3 methods for POS Tagging were implemented and a detailed analysis was done. We further made a comparison of them on various parameters. The difficulties for implementation of each algo were overcome and a fair enough accuracy was achieved in all 3 methods. We concluded that HMM gives highest accuracy on our brown corpus dataset, however, with proper feature engineering in SVM and proper selection of hyperparamters in Bi-LSTM, we suspect that they can achieve much higher accuracy.