Assignment 1: POS

Inderjeet Jayakumar Nair, 170020013
Saurabh Jayesh Parekh, 170100016
Manas Jain, 170040068

September 2020

1 Overall Accuracy

We employed Five-Fold cross validation evaluation to compute the mean accuracy. The accuracy in every fold is defined to be number of correctly tagged words upon the total number of examples. The results are given in the table below.

Model	Accuracy Test (%)	Accuracy Train (%)
HMM	96.01	97.35
SVM	83.25	83.36
Bi-LSTM-baseline*	79.38	78.39
Bi-LSTM-CNN*[2]	87.15	87.19

Table 1: Accuracy for different models

We performed our experiments with two variants of Bi-LSTM. In one of the models(baseline), we only fed the RNN model with the word embeddings of the tokens. In the CNN variant[2], we constructed a network that leverages letter and inter-letter relations and encodes it to a dense vector. Thereafter, the dense representation is concatenated with the word embeddings of the tokens and these features are fed into the Bi-LSTM model.

2 Confusion Matrix

The confusion matrix is a square matrix where $[i, j]$ element denotes the number of i^{th} tag assigned to the instances whose true label is the j^{th} tag. The confusion matrices for various models are shown in this section.

2.1 Hidden Markov Model

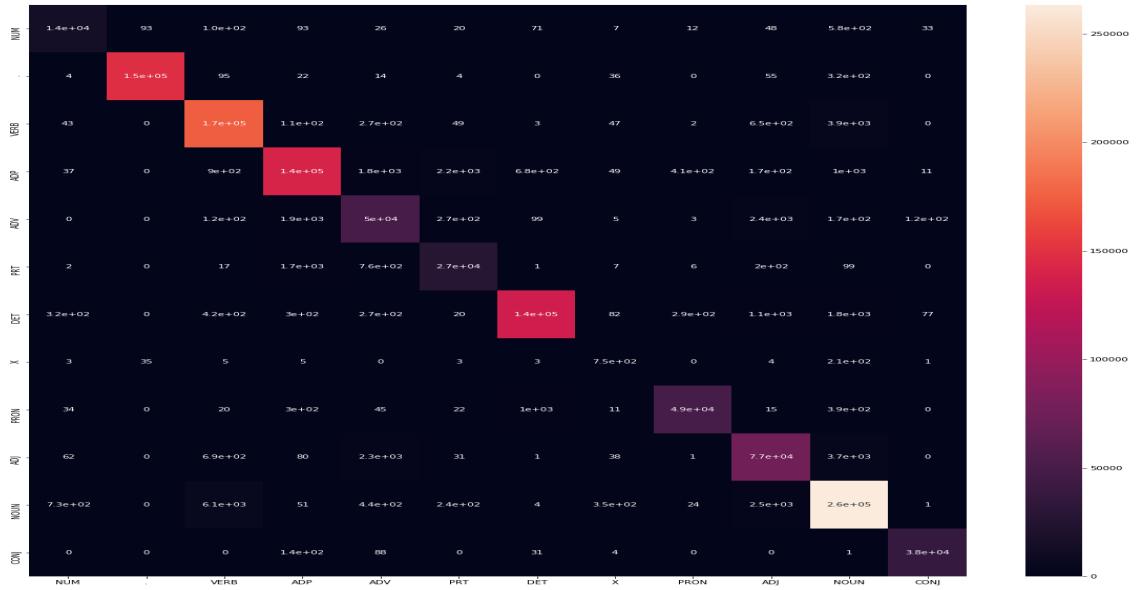


Figure 1: Heat map of the confusion matrix obtained from HMM

	NUM	.	VERB	ADP	ADV	PRT	DET	X	PRON	ADJ	NOUN	CONJ
NUM	13641	93	105	93	26	20	71	7	12	48	576	33
.	4	147437	95	22	14	4	0	36	0	55	316	0
VERB	43	0	174275	108	272	49	3	47	2	652	3901	0
ADP	37	0	900	140003	1777	2234	684	49	412	174	1020	11
ADV	0	0	115	1921	50269	271	99	5	3	2384	172	123
PRT	2	0	17	1748	760	26936	1	7	6	197	99	0
DET	320	0	421	296	266	20	135090	82	292	1107	1840	77
X	3	35	5	5	0	3	3	753	0	4	214	1
PRON	34	0	20	295	45	22	1032	11	48582	15	393	0
ADJ	62	0	690	80	2279	31	1	38	1	76627	3689	0
NOUN	728	0	6107	51	443	239	4	347	24	2458	263337	1
CONJ	0	0	0	144	88	0	31	4	0	0	1	37905

2.2 Support Vector Machine

Figure 2: Heat map of the confusion matrix obtained from SVM

	NOUN	VERB	ADP	.	DET	ADJ	ADV	CONJ	PRON	PRT	NUM	X
NOUN	224965	21701	2019	0	456	23220	4337	72	360	2666	425	683
VERB	18200	138233	4693	0	806	6034	6399	86	339	3233	132	82
ADP	3779	4664	130008	4	1233	1902	2889	69	840	8204	14	22
.	87	189	140	147557	44	4	50	1	0	5	3	25
DET	1002	900	2053	1	132603	1412	1394	58	1472	482	247	20
ADJ	19266	6032	949	0	500	45663	3444	67	19	560	149	75
ADV	4531	8120	2447	0	520	4935	36337	227	99	1093	9	9
CONJ	364	570	174	1	393	143	375	37518	27	11	7	5
PRON	615	751	699	2	446	42	398	12	46129	178	13	13
PRT	1444	1511	1558	0	15	197	592	34	40	13368	9	10
NUM	988	55	18	0	3	114	16	5	0	10	13864	7
X	317	24	8	0	0	55	8	2	9	19	2	435

2.3 Bi-LSTM Baseline

Figure 3: Heat map of the confusion matrix obtained from Bi-LSTM Baseline

Figure 4: Heat map of the confusion matrix obtained from CNN Bi-LSTM

3 Per POS Accuracy

We compute the per-POS precision, recall and F1-score from the confusion matrix.

3.1 Hidden Markov Model

Tag	Precision	Recall	F1-Score
NUM	0.926384	0.917104	0.92172
.	0.99631	0.999133	0.997719
VERB	0.971693	0.953625	0.962574
ADP	0.950455	0.967099	0.958705
ADV	0.908005	0.893846	0.90087
PRT	0.904712	0.903014	0.903862
DET	0.966233	0.985922	0.975978
X	0.733918	0.54329	0.624378
PRON	0.962992	0.984757	0.973753
ADJ	0.917711	0.915266	0.916487
NOUN	0.962	0.95565	0.958815
CONJ	0.992979	0.993552	0.993266

Table 2: Hidden Markov Model Per POS accuracy

3.2 Support Vector Machine

Tag	Precision	Recall	F1-Score
NOUN	0.800860	0.816398	0.808555
VERB	0.775557	0.756405	0.765861
ADP	0.846252	0.898056	0.871385
.	0.996299	0.999946	0.998119
DET	0.936171	0.967771	0.951708
ADJ	0.595159	0.545419	0.569204
ADV	0.622987	0.646117	0.634342
CONJ	0.947711	0.983408	0.96523
PRON	0.935717	0.935035	0.935376
PRT	0.711897	0.448154	0.550044
NUM	0.919363	0.932096	0.925686
X	0.494881	0.313853	0.384106

Table 3: Support Vector Machine Per POS accuracy

3.3 Bi-LSTM

Tag	P(Baseline)	R(Baseline)	F1(Baseline)	P	R	F1
NOUN	0.872538	0.859111	0.865772	0.8467462957	0.8749519157	0.8606180673
VERB	0.925502	0.914175	0.919804	0.8455539818	0.859124487	0.8522852188
.	0.612080	0.981999	0.754118	0.992072625	0.9990241588	0.9955362569
ADP	0.691465	0.697326	0.694383	0.9009276846	0.9043007336	0.9026110579
DET	0.916463	0.795872	0.851921	0.9399036033	0.9507075661	0.9452747149
ADJ	0.747382	0.712002	0.729263	0.704576745	0.605152829	0.6510910633
ADV	0.686423	0.731869	0.708418	0.6817838382	0.6738917833	0.677814839
PRON	0.959272	0.933033	0.945971	0.8935800928	0.9287915028	0.9108456248
CONJ	1.000000	0.027419	0.053375	0.9679943101	0.9631988677	0.965590635
PRT	0.855478	0.058729	0.109913	0.7969689423	0.7510141138	0.7733093997
NUM	1.000000	0.112994	0.203046	0.8789575866	0.7913809332	0.8328734168
X	NaN	0.000000	NaN	0.6857142857	0.0173160173	0.0337790289

Table 4: Bi-LSTM Baseline and CNN Per POS accuracy

4 Strength and Weakness of the model

4.1 Hidden Markov Model

4.1.1 Strengths

1. **Per POS-tags Performance:** When presented with a sufficient number of examples for a particular POS-tags, the per POS accuracy for that tag is high. This is illustrated in 5. We can see from the table the per POS accuracy of '.', 'VERB', 'ADP', 'DET', 'CONJ', etc is very high as the number of examples corresponding to each of them is significant.

2. **Algorithmic Efficiency:** This approach is extremely fast in terms of training. The training stage essentially involves computation of transition probabilities and emission probabilities. The asymptotic complexity for the construction of the transition or emission table in our implementation was $\mathcal{O}(n)$, where n is the number of tagged examples available in the training set.

3. **Applications:** The generality of this model allows itself to be used for various other sequence labelling tasks such as decoding of electrical signals etc. The decoding algorithm is well solved problem which further enhances its applicability.

4.1.2 Weakness

1. **Per POS-tags Performance:** The model does not perform well when provided with a very small number of examples. The reason is learning from a set of tags from the training examples is not useful towards unseen combination of tags as the information obtained from it is not transferable to other tags not present in the examples. Thus performance is severly affected if not enough examples are made available for all the tags. The accuracy of 'X' is very low as very few examples of it are present in the corpus.

2. **Label Bias Problem:** The inference of tags is vastly affected from the problem of label bias[1]. The problem arises due to the fact that the emission probabilities from every tag must sum to one. Thus the edges emanating from states or tags having fewer number of outgoing edges may be associated with higher probability. Thus there is an inherent bias towards choosing the tags which contain lower number of outgoing edges.

3. **Vitterbi Decoding Complexity:** The time complexity for decoding is of the order of $\mathcal{O}(k^2)$ which increases very rapidly for corpora containing large cardinality of tagset such as Brown Corpus. For example, the brown corpus has over 400 tags and decoding a sentence of under 100 tokens takes around a second. Thus, when presented with a large number of tags, the method becomes highly unscalable when presented with documents containing large number of sentences.

4. **No scope for feature engineering:** The HMM model for POS tagging implicitly assumes that the distribution followed by the tags is multinomial in nature. Thus the maximum likelihood estimation of parameters does not permits the integration of features that might be useful for increasing the accuracy. This is also disadvantageous for the computation of the probability of an unseen word given a tag.

5. **Markov Assumption:** The model simplifies the hypothesis function by assuming that the probability of future tag is conditionally independent of past tags given the present n tags. It is well known that POS tagging contains examples showing wide range dependencies between tags and the dependence of a present tag with one in future.

4.2 Support Vector Machine

4.2.1 Strengths

1. **Feature Engineering:** The SVM model takes different features as input and trains the weights by maximizing the margin so as to give importance to features which can better classify the tags by assigning them higher weights. So it is easy to improve the accuracy if one can choose the appropriate features for the same.

2. **Word-Embeddings:** Using pre-trained word embeddings of current word, previous words and following words can drastically improve the accuracy of the model.

3. **Applications:** It can be used in almost all applications where Artifical Neural Networks are used like Handwriting Recognition, Intrusion Detection, Protein Structure Prediction, etc. Also SVMs give better results than Artifical Neural Networks as they have generalizations in practice.

4. **Model Explainability:** Unlike Bi-LSTM, it is relatively much easier to explain the results based on the features given as the input. One can predict the changes in the accuracy due to a particular feature by appropriate feature selection and engineering.

4.2.2 Weakness

1. **Long Training Time:** Since the problem of PoS tagging is a multi-class classification problem, the time complexity of training is atleast O(K) even if most efficient algorithms are implemented. So it takes significantly higher time for training the model.

2. **Visualisation not feasible:** Since the data-points are n-dimensional feature vectors and also the classification hyperplane, so it is very hard to visualise and interpret the final model. One needs to perform dimensionality reduction to 2-d to be able to visualise the plane but it leads to loss of data.

3. **Hyperparameter-Tuning difficult:** It is difficult to interpret the impact of the hyperparameters like Cost and learning rate. Also it doesn't improve the accuracy much even after doing it for hours.

4. **High Memory Requirement:** As the number of features required for training the SVM model can be significantly high especially if word embeddings are used, it consumes a lot of RAM for large data sets.

5. **Exclusion of Probabilistic Approach:** The SVM classifier separates the data points below and above the classifying hyperplane. So it does not take into account the probabilities of the classes. There is no probabilistic explanation as such for the classification.

4.3 Bi-LSTM

4.3.1 Strengths

1. No feature engineering is required for training the model in deep learning. Using the LSTM layer does all the part understanding the context in the sentence and hence we are able to achieve high accuracy using this model.

2. Here we have not used the Markov assumption as done in the HMM part rather using Bi-LSTM, our model at a particular time takes into account context from both direction i.e. words occuring before and the words which are ahead in the sentence

3. Instead of manually creating features as done in SVM, here in order to get semantics of the words in the sentence we have used pre-trained word2vec (by Google) and GLoVe embeddings (by Stanford) which act in some way a feature vector in our deep learning model

4.3.2 Weakness

1. Deep Learning models are data hungry and with more data chances of accuracy improvement is higher as compared to other machine learning and probabilistic models

2. Highly computation heavy, requires very high RAM and disk space for the training procedure. Also the training is time consuming. There were instances in training where if we increased the model complexity the server crashed due to high RAM usage so we had to choose a less complex model

3. Explainibility is a problem in this model. It is not intuitive to know what actually the model is learning and so it is not easy to explain this model correctness due to the black-box nature of deep learning as a whole.

5 Detailed Error analysis

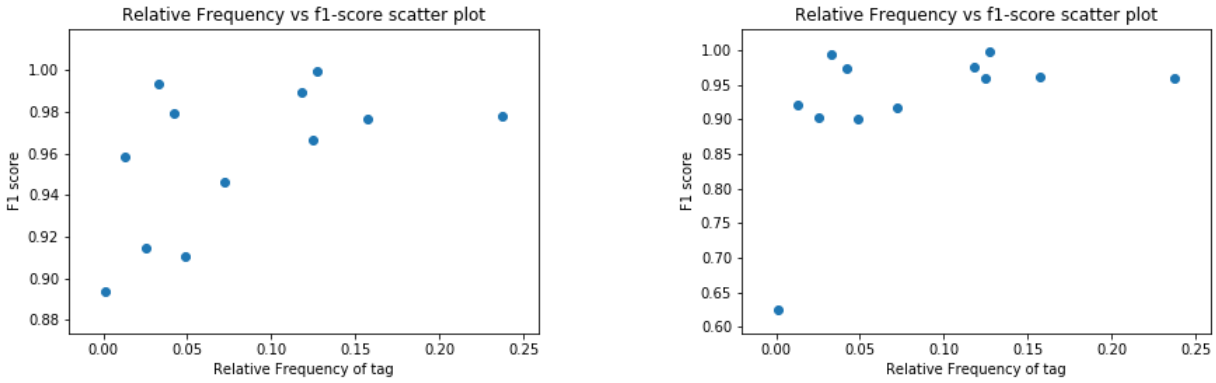
5.1 Hidden Markov Model

Referring the overall accuracy in 1, we see that the HMM's test and training accuracy is nearly equal and the error is very close to Bayes' error. Thus our model is well-fitted under this hypothesis. We present a table below which contains the per-pos accuracy of the model over the training and testing examples with the relative occurrence frequency of the tags. This will help in illustrating the performance of model learned on biased data over minority class datapoints.

Tag	P(Train)	R(Train)	F1(Train)	P(Test)	R(Test)	F1(Test)	Relative Frequency
NUM	0.936041	0.982217	0.958574	0.926384	0.917104	0.92172	0.013
.	0.999729	0.999226	0.999477	0.99631	0.999133	0.997719	0.127
VERB	0.983274	0.97044	0.976815	0.971693	0.953625	0.962574	0.157
ADP	0.964946	0.967964	0.966453	0.950455	0.967099	0.958705	0.125
ADV	0.910914	0.91044	0.910677	0.908005	0.893846	0.90087	0.048
PRT	0.912382	0.916809	0.91459	0.904712	0.903014	0.903862	0.026
DET	0.992519	0.986168	0.989333	0.966233	0.985922	0.975978	0.118
X	0.89032	0.897547	0.893919	0.733918	0.54329	0.624378	0.001
PRON	0.972984	0.985912	0.979406	0.962992	0.984757	0.973753	0.042
ADJ	0.94089	0.951676	0.946252	0.917711	0.915266	0.916487	0.072
NOUN	0.97737	0.978736	0.978053	0.962	0.95565	0.958815	0.237
CONJ	0.993058	0.99365	0.993354	0.992979	0.993552	0.993266	0.033

Table 5: Per POS accuracy and Relative Frequency of tags

We note from the table that the per-POS accuracy decreases with the relative frequency. The emission probabilities and transition probabilities are estimated from the corpus containing examples of relevant tag and word combinations. If the number of examples are very less for the joint occurrence of a word and a tag or bi-gram occurrence of tags then the joint probabilities estimated for them will have high probability of being invalid. The features shared between different bi-gram tag combinations is almost null, thus this model cannot be readily generalized for examples containing tags from the minority class. The 'X' and 'ADV' tag are associated with lowest train and test per-POS Tag F1-Score with the relative occurrence being minimum for 'X' and 5^{th} minimum for 'ADV'. We also see that 'NOUN' and 'DET' is associated with very high accuracy as their relative occurrence is also high. We demonstrate the relation between relative occurrence and per-POS F1-Score for every tag in the below plots.

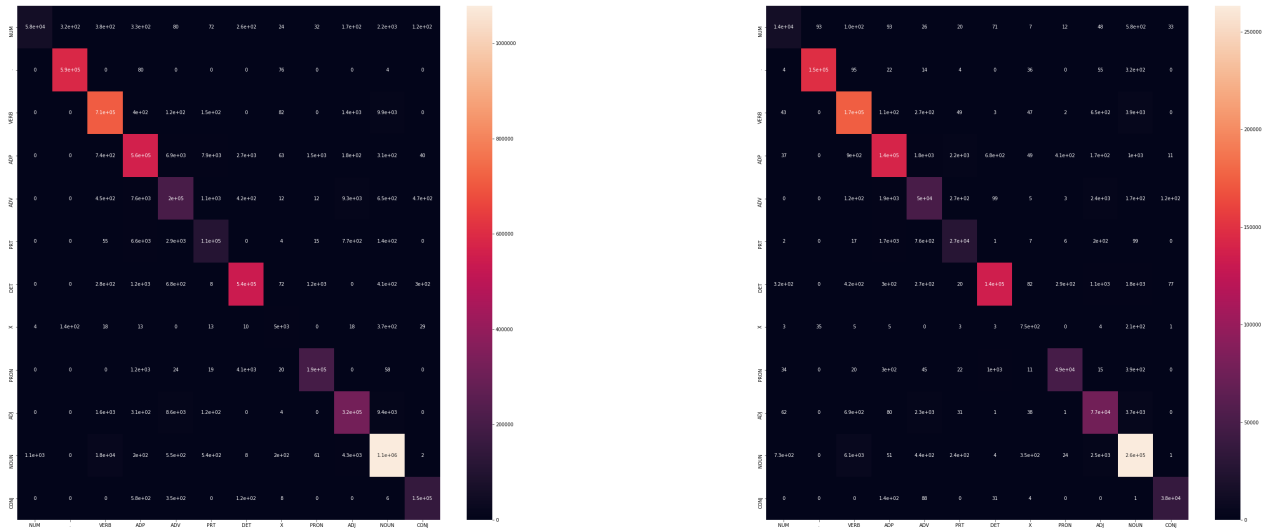


(a) Train Set

(b) Test Set

Figure 5: Per POS Tag F1-Score sv Relative Occurrence

The heat-map of the confusion matrix between the tags is shown below for the training and testing set.



(a) Train Set

(b) Test Set

Figure 6: Heat Map of Confusion Matrix

The presence of the bright rectangles along the rectangles signifies the accuracy of the model. To get the perfect results, each of the heat-maps in 9 must have pitch black intensities at the non-diagonal elements. Refer to the figure 1 for the subsequent discussions. From the image, we see that a lot of examples of 'VERB', 'ADV', 'PRT', 'NOUN' and 'DET' are classified as 'ADP'. Also many examples of 'ADP' are classified as 'PRON', 'DET', 'PRT', 'ADV' and 'VERB'. Thus the training examples for the 'ADV' were sufficiently 'confusing' for the model to learn to derive correct predictions.

We now set to explain why the model gives poor accuracy for 'ADV' in spite of tonnes of training examples. We believe that the training examples containing 'ADV' is highly irregular and unpredictable. We need a means to quantify this uncertainty with respect to a tag. We associate this uncertainty with the entropy of transition probability distribution from a tag. If the entropy is high, then there is a very high chance of unpredictable transitions from the given tag. We report some of the entropy values for the tag transition probability distribution along with its performance

Tag	$H(\mathrm{P}(.\|\mathrm{Tag}))$	F1-Score
ADV	2.10	0.900
PRON	1.14	0.973
ADJ	**1.28**	**0.916**
CONJ	**2.04**	**0.993**

Table 6: Some Tags with the transition distribution entropy and F1-Score

We note that some of the examples are not complacent with our hypothesis. We believe that entropy alone does not decide the performance with respect to a particular tag. We must also look into the relative frequency of occurrence f. Let us define a quantity $M(S) = \log(\frac{H(\mathrm{P}(.|S))}{f})$. If this value is high, then the entropy is high and/or the relative frequency is low. This would be a plausible reason for bad performance of tags having high M. On the contrary, if M is low, then the uncertainty associated with the tag is very small and/or the relative frequency is high. This case is fruitful for having high performance of the tags. The following table confirm our hypothesis.

M range ($M(S) = \log(\frac{H(\mathrm{P}(.\|S))}{f})$)	Average F1-Score
0.5 - 1	0.967
1 - 1.5	0.962
1.5 - 2	0.933
2 - 2.5	0.922
3 - 3.5	0.624

Table 7: Explain-ability of per-POS performance by using $M(S) = \log(\frac{H(\mathrm{P}(.|S))}{f})$

We performed a hyperparameter optimization for increasing the accuracy of the model. Our model resorted to zero order back off to compute the probabilities of unseen words given any tag and for the unknown instances of transitions. For unseen instances, we hypothesized

$$P(s_i|s_j) = \lambda \textbf{ and } P(w_i|s_j) = \lambda$$

where w_i is an unseen word and $s_i|s_j$ is an unseen instance of transition. We obtain the following accuracy for different values of λ.

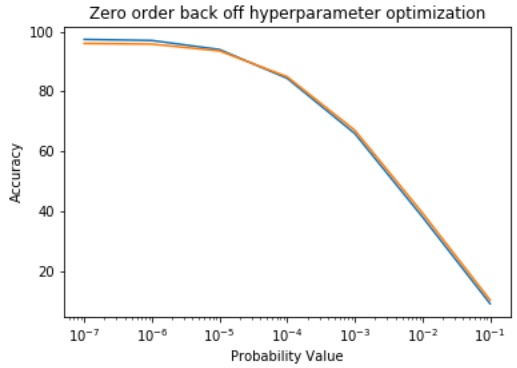


Figure 7: Hyperparameter optimization for zero order back off

Thus we set the value of $\lambda = 10^{-7}$. The orange curve in 7 corresponds to the test accuracy and the blue curve corresponds to train accuracy.

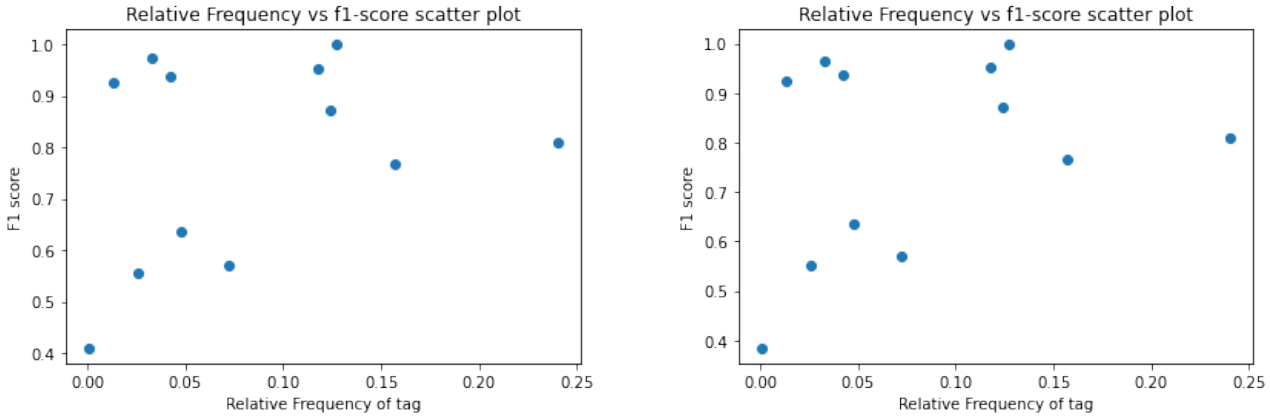
5.2 Support Vector Machine

Initially using only the word length, capitalisation, upper-case and lower-case features gave very less accuracy of about **40%** only. After adding the prefix and suffix features also, the accuracy jumped close to **55%** indicating that prefix and suffix play a significant role in PoS tagging. Adding few more features like stem word using PorterStemmer, isNumeric or not, tag of previous word,etc and doing some normalization further increased the accuracy upto **65%**. When the pre-trained Glove word-embeddings consisting of 1 lakh 50-dimensional embeddings was used, the accuracy significantly improved to about **83%**. Using the pre-trained word2vec 300-dimensional embeddings trained from GoogleNews dataset consisting of 100 million unique words, the accuracy increased upto **88%**. Further taking the embeddings of previous 3 words and following 3 words would increase the accuracy upto **95%**. It was observed that in general including previous 3 words features and next 3 words features gave the best accuracy.

The table indicating the accuracies for different features selected is shown below.

Features Selected	Accuracy
Word length, capitalisation, upper-case, lower-case, isNumeric	40%
Prefix and suffix for nouns, verbs, adjectives and adverbs	55%
Word stems using PorterStemmer	60%
Tag of Previous Word	65%
Pre-trained Glove word-embeddings (1 lakh 50-dimensional vectors)	83%
Pre-trained word2vec embeddings (10 crore 300-dimensional vectors)	90%
Including the features of previous 3 words and following 3 words	95%

Relation between relative occurrence and per-POS F1-Score for each tag is shown below plots.

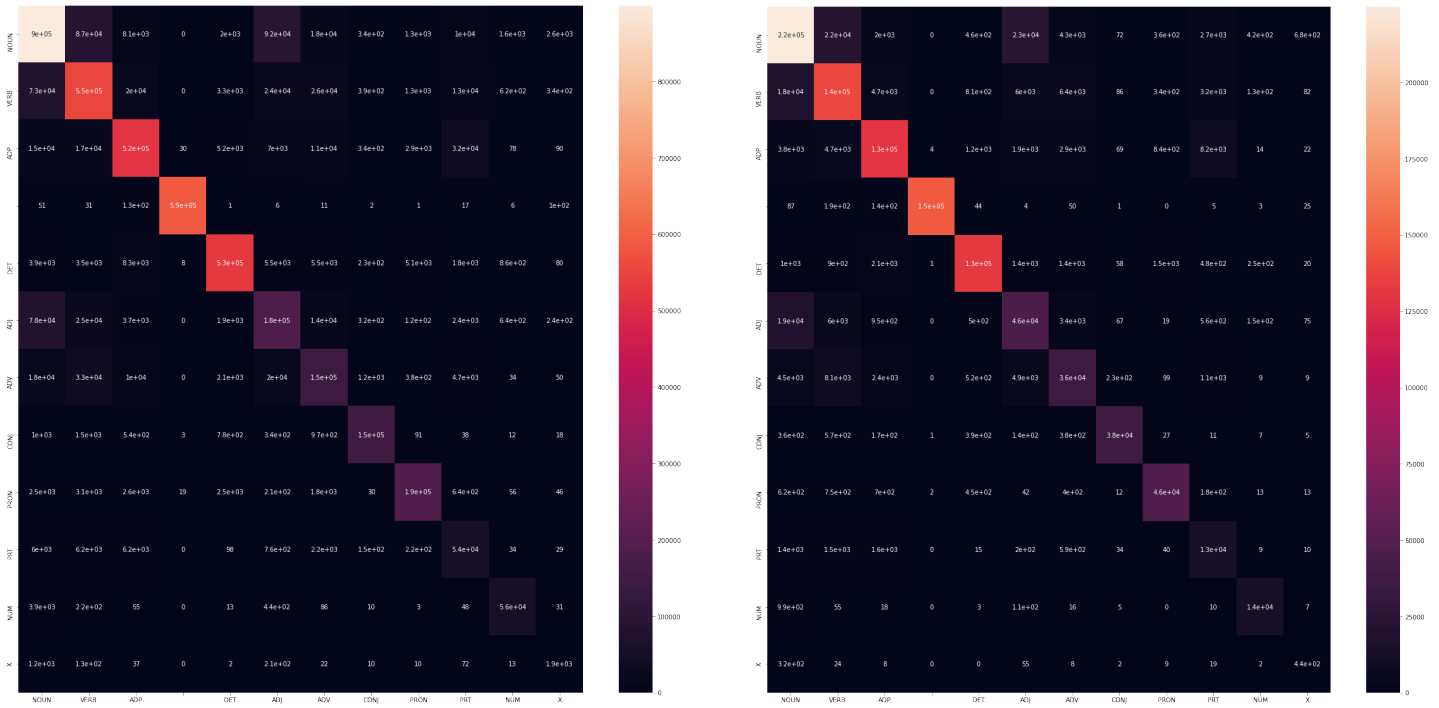


(a) Train Set

(b) Test Set

Figure 8: Per POS Tag F1-Score sv Relative Occurrence

The heat-map of the confusion matrix between the tags is shown below for the training and testing set.



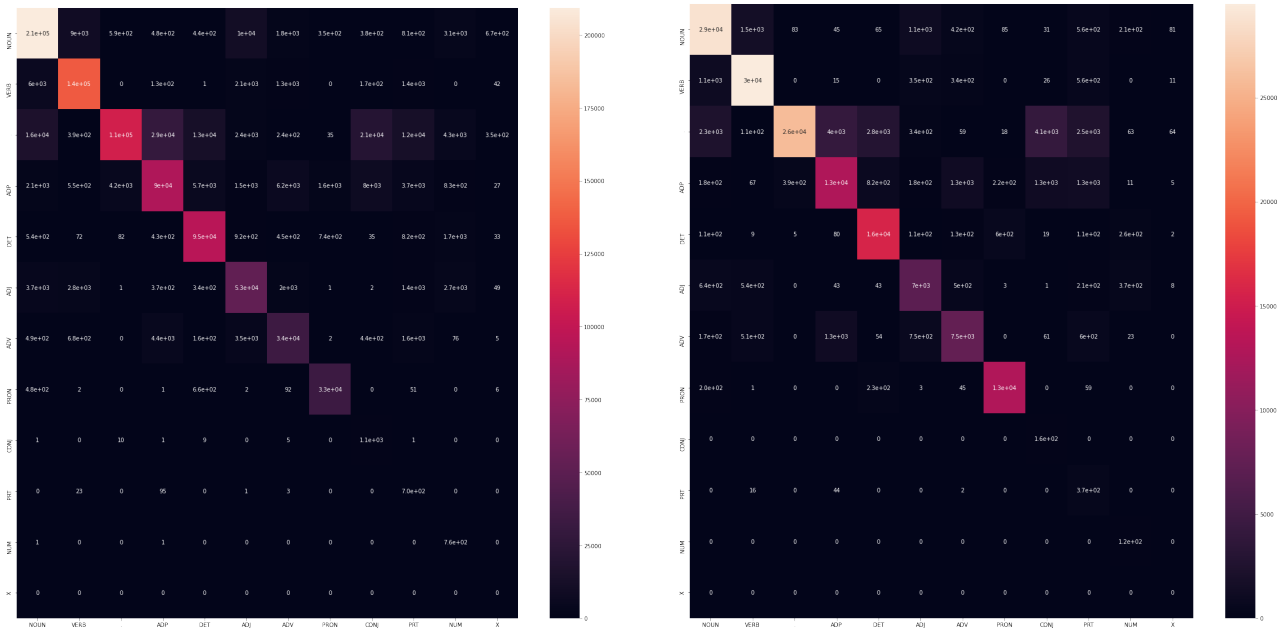
(a) Train Set

(b) Test Set

Figure 9: Heat Map of Confusion Matrix

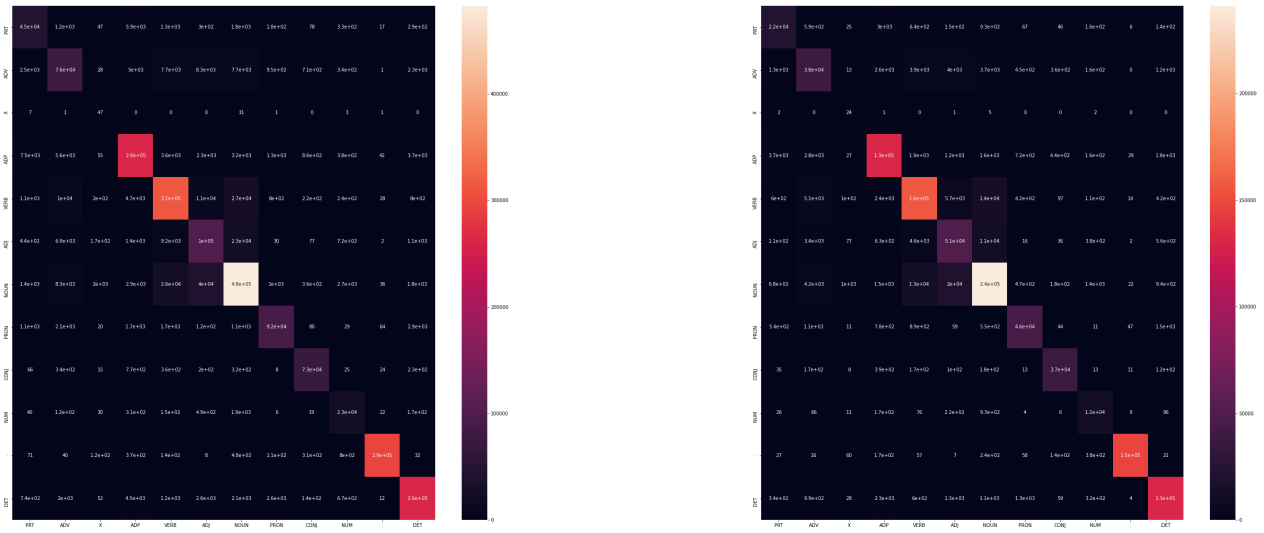
5.3 Bidirectional Long Short Term Memory (Bi-LSTM)

We had the liberty of implementing a couple of models, namely: Bi-LSTM baseline which only uses the word embeddings and a variant of Bi-LSTM which uses the letter level information along with the word level information to assign labels. For the second model, we construct a trainable dictionary of letter embeddings and capture the letter level and inter letter level information from the tokens using Convolutional Neural Networks. The word embeddings is derived by following the distributional hypothesis and is expected to contain very less information about the letter states. POS tagging relies heavily on letter level information such as prefixes' and suffixes and the use of CNNs can extract these without the need for manual feature engineering. We obtained an accuracy of **78 %** for the baseline model and about **87 %** for the bi-LSTM CNN model. The number of RNN units required were also very less compared to that of the baseline model. We show the heatmaps for both the models' confusion matrix over testing and training set.



(a) Train Set (b) Test Set

Figure 10: Heat Map of Confusion Matrix for the baseline Bi-LSTM

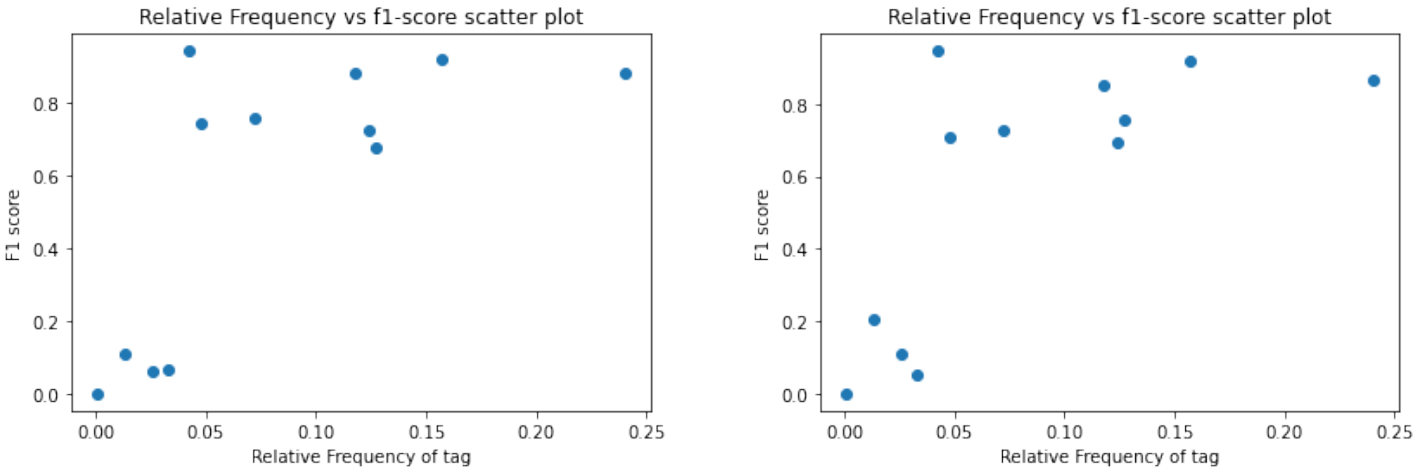


(a) Train Set (b) Test Set

Figure 11: Heat Map of Confusion Matrix for the CNN Bi-LSTM

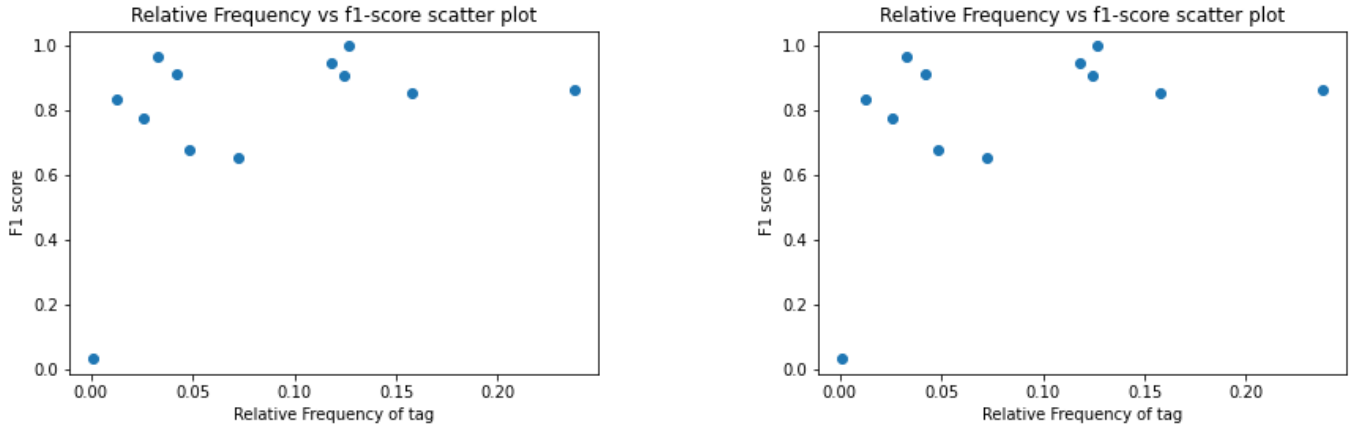
Deep Learning is a technique which is highly data driven and works very well on problems having a huge corpus of data. Here since we modelled the problem as a multi-class sequence labelling where labels corresponds to the POS tags of the words in a sequence. In the brown corpus we have around 57000 number of sentences with maximum number of words of 180. We have used Universal tagset of POS tags which contain 12 most commonly used tags. And so we have a mismatch in the number of examples of a particular labels which led to our model not able to correctly predict the labels having less examples as compared with the one having various examples in the training set. For example this model is very bad on predicting the POS tag "X" because of having very

examples in the corpus whereas for POS tags like "VERB" and "PRON" the model has a very good F1 score.



(a) Train Set (b) Test Set

Figure 12: Scatter plot for per POS accuracy vs Relative Frequency for the baseline Bi-LSTM



(a) Train Set (b) Test Set

Figure 13: Scatter plot for per POS accuracy vs Relative Frequency for the CNN Bi-LSTM

To remove such bias from our model the dataset can be made more uniform by adding examples such that the count mismatch can be reduced. This point is also clear by the scatter plot where the F1 score is higher for the POS tags having high relative frequency in the data corpus.

Also since we are padding every sentence to a MAX-SEQUENCE-LENGTH of 180 for feeding it as same dimesional input which is very useful for training the model. The model becomes biased to predict pad label because of seeing huge examples having a pad as a word. But this is not right because pad is a label we have added manually and should not be counted to the total accuracy. We manually removed the label for the pad and achieved around 80% accuracy on training the model for 3 epochs and 16 BiLSTM units.

6 Learning

6.1 Hidden Markov Model

Through this assignment, we learnt how to implement a graphical generative model in a programming language. The power of this method is realized from the five-fold evaluation output. This sub part was also helpful in making

us understand the necessity of efficient algorithm for a certain task by reducing the time complexity to polynomial from exponential. Detailed error analysis helped us in looking at the model at a variety of aspects in scrutiny. We got introduced to a evaluation metric for assessing the performance of the model by implementing the computation of confusion matrices

6.2 Support Vector Machine

This assignment helped us in learning the implementation of Support Vector Machine as well as using it for implementing the PoS Tagger. We learned a lot about feature selection and engineering and it's importance in training SVM. We also learned about various evaluation metrics like F1-score, precision, recall and confusion metrics. We also plotted scatter plots and heat maps of the confusion matrices to get a better understanding of SVM performance. We learned about the strengths and weaknesses of the HMM, SVM and Bi-LSTM models.

6.3 Bidirectional Long Short term Memory (Bi-LSTM)

We learned to implement POS tagger using deep learning using LSTM and pretrained embeddings. We learned about Pytorch, Keras framework required to implement different layers in the architecture. We also performed 5 fold cross validation required for better analysis of the model metrics and its dependence with different train and test data. Also we analysed confusion matrix and per POS accuracy and F1 scores. Also we learned about hyperparamameter tuning by doing various experiments involving the analysis of model behaviour on changing LSTM neurons, batch size, learning rate, embedding dimension which also explained computational resource constraints and use of GPUs. Having this deep learning part helped us in understanding the cutting edge techniques used in the field of NLP.

References

[1] Léon Bottou. "UNE APPROCHE THEORIQUE DE L'APPRENTISSAGE CONNEXIONNISTE ET APPLICATIONS A LA RECONNAISSANCE DE LA PAROLE". PhD thesis. 1991.

[2] Xuezhe Ma and Eduard Hovy. "End-to-end sequence labeling via bi-directional lstm-cnns-crf". In: (2016).