MVA Project Report
Part Of Speech Tagging

Group 8
Sannidhya Basu - MB2103
Prasun De - MB2117
Nitin Prasad - MB2132

February 4, 2022

Abstract

We discuss the famous Part Of Speech Tagging problem and view it as a Classification Problem and then as a sequence to sequence labelling problem. We apply some of the most well - known techniques to solve those problems on a data-set, and report our results.

1 Parts of Speech (POS)

Merriam Webster [3] defines Parts of Speech as a traditional class of words (such as adjectives, adverbs, nouns, and verbs) distinguished according to the kind of idea denoted and the function performed in a sentence.

Parts of speech (**POS**) are useful clues to sentence structure and meaning. We are interested in the task of **POS tagging**. Via POS tagging, we are extracting some useful information from raw text data ! The extracted POS tags may be used in a plethora of other linguistic tasks like Sentiment Analysis, etc. Hence, they are relevant in NLP.

2 POS tagging as Classification Problem

We begin by attacking the problem purely from a classification point of view. We assume that we can represent the words in a feature-space of dimension L, and hence we have data of the form $\{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathbb{R}^L$ for $i = 1, \cdots, n$. Note that y_i's are the labels and they correspond to the POS-tag of x_i .

We try a variety of classifiers on the data which we have described in the **Data Analysis section** . We chose to present the SVM approach in greater detail because it has been used by [6] , [4] with success on very large data-sets.

2.1 SVM based approach

We follow Nakagawa et.al. [6] and use the Support Vector Machine (SVM) introduced by Vapnik [1] to build a classifier.

We recall the SVM setup now. For binary classification : We have training data $\{(x_i, y_i) | x_i \in \mathbb{R}^L, y_i \in \{+1, -1\}, 1 \leq i \leq l\}$.

$$w^t x + b = 0, \text{ where } w \in \mathbb{R}^L \text{ , } b \in \mathbb{R} \tag{1}$$

We assume that the hyperplane (1) is a separator of the training data into the two classes such that

$$y_i \cdot (w^t x + b) \geq 1 \tag{2}$$

SVMs find the optimal hyperplane that maximizes the margin (the distance between the hyperplane and the nearest points). Formally, we aim to minimize the following loss function (soft margin SVM)

$$\min_{w,b} \frac{1}{2} w^t W + C \sum_{i=1}^{l} \max(0, 1 - y_i(w^t x_i + b))$$

where C is the regularization parameter that controls the strength of the penalty.

Now we will tackle the original multi-class classification problem by the **one-versus-rest approach**. So, we train k classifiers $h_1, \cdots, h_k$ such that

$$\begin{cases} h_i(x) \geq +1 \text{ if } x_i \in \text{ class } i \\ h_i(x) \leq -1 \text{ else} \end{cases}$$

for $i = 1, \cdots, k$. We observe that in this problem formulation, there is an utmost importance of the feature-extraction that we are doing.

2.2 Feature Extraction

We assume that the classifier will be passed untagged sentences during training . We use the following ideas to construct the feature map ([6]) :

1. Features in Context of POS : This includes the (important) information about the tags of the previous and preceding words. However, we note that we have assumed that our classifier will receive test data from untagged sentences. So, there isn't any straightforward method to include all these tags in our feature representation.
2. Features in Context of word : This includes features like the preceding and succeeding words of the target word. We shouldn't forget that the target word itself is important, so we include it as well. For example, **the** is a determiner.
3. Features in Context of sub-strings of word : We can include suffixes/prefixes of the word because that helps us capture important information about the word. For example,
   - Prefix **un-** might suggest that it is an adjective eg; underweight.
   - Suffix **-ed** points us to a past tense verb eg; appreciated.
   - Suffix **-ly** leads us to an adverb eg; adventurously.
   - A **hyphen** in the word might mean an adjective , eg; multi-disciplinary.
   - Capitalization might point to proper noun eg; Microsoft.

2.3 Representation of features

We perform **one-hot-encoding** on the word features to convert it into vectors in $\mathbb{R}^L$ for some L which will then allow us to apply the classification algorithm.

We observe that this type of feature representation does bring forth some sense of redundancy. Moreover, the **length of the feature vector will increase linearly** with the number of distinct words in the training corpus. So we realize that we need to put down **finite vocabulary** assumption. Needless to say, it is a challenge to handle unknown words that the classifier encounters on the test data.

2.4 Pros/Cons of classification based approach

1. **Pros :**
   - Intuitive, easy to understand and implement.
2. **Cons :**
   - Disregards the intricate inter-dependence amongst the tags in the sentence. For example, consider the **preposition rule** : **Rule**: A preposition is never followed by a verb. As we are not taking the POS context into consideration, our hope is that the classifier learns this on its own !

   HMM to better capture these ideas.

We observe that a natural extension of a single word POS labelling will be to consider the the sequence of labels of all the words in the sentence. We now move onto viewing **POS tagging as a sequence to sequence labelling** problem now.

3 POS tagging as a sequence to sequence labelling Problem

In the earlier section we looked at **POS** tagging as a **Classification** problem. In this section, we look at it from a different Probabilistic aspect. To that end, we introduce the **Hidden Markov Model**. We have based our approach on Jurafsky [2], though our notations are a bit different.

3.1 Hidden Markov Model Approach

Notation:

- $1:n = \{1, 2, \cdots, n\}$
- $x_{1:n} = (x_1, x_2, \cdots, x_n)$

We all are familiar with a Markov Chain. Every Markov Chain $\{X_t\}_{t\in\mathcal{S}}$ has 3 components:

- $\mathcal{X} = \{x_1, x_2, \cdots x_N\}$ be the sample space (finite in our case)
- $\mathbf{A}$ = A transition probability matrix which is a stochastic matrix.
- π = an initial distribution (not necessarily stationary).

A Hidden Markov Model has some extra structure to it.
Definition : Let X_t and Y_t be discrete time stochastic processes. Then the pair (Y_t, X_t) is a Hidden Markov Chain if:

- Y_t is a Markov Chain whose behaviour is not directly observable.
- $\mathbb{P}[X_t \in A|Y_1 = y_1, ..., Y_t = y_t] = \mathbb{P}[X_t \in A|Y_t = y_t] \quad \forall \quad t \geq 1$, for certain "good" sets A.

A **HMM** has a bit of additional structure :

- $\mathcal{X} = \{w_1, x_2 \cdots, w_M\}$ is the all possible values of observed variable.
- $\mathcal{Y} = \{s_1, s_2, \cdots, s_N\}$ is the all possible values of hidden variable.
- π = an initial distribution (not necessarily stationary), for the unobserved variable.
- $\mathbf{A} = ((a_{ij}))$ where $a_{ij} = \mathbb{P}(Y_{t+1} = s_j|Y_t = s_i)$. This is transition probability matrix for Y which is a stochastic matrix.
- $\mathbf{B} = ((b_{ij}))$ where $b_{ij} = b_{s_i}(w_j) = \mathbb{P}(X_t = w_j|Y_t = s_i)$. This is also called emission probabilities.

As we see, a Hidden Markov Model has some extra structure to it.
Now we describe how we apply this hidden Markov Model for the part of speech tagging problem. We will denote an **HMM** as $H \equiv (\mathcal{X}, \mathcal{Y}, A, B, \pi)$. We denote the model parameters as $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$ where $\mathbf{A}$ and $\mathbf{B}$ are the matrices defined above.
Suppose we have a sentence where we denote the constituent words as $\mathbf{x} = x_1, x_2, \ldots x_T$. Here we (the machine) don't (doesn't) have any idea about the tags of the words of the sentence, or in other words the tags can be thought of as unobserved states. And for a given tag, the probability of each of the words, give us the B matrix. For POS tagging, we assume that the tag of a word only depends on the tag of the word immediately before it, i.e the Markov condition. So, given a sequence of words, we desire to find a sequence of tags $\mathbf{y} = y_1, y_2, \ldots, y_T$ which is in some sense "best" for this sequence of words.
For the time being, we assume that we have some estimate of the entries of the A and B matrix. This is not always the case, we might need to estimate the entries of those matrices, and this is discussed in the next section, where we use the Baum Welch Algorithm, which is a sort of **EM Algorithm**, to estimate the entries of A and B given some initial guess of the estimates.
So given the word sequence $\mathbf{x}$ and the matrices A and B, it is reasonable to find a tag sequence which maximises

$$\mathbb{P}(y_1, \ldots, y_T|x_1, \ldots, x_T)$$

Thus we have framed our problem as

$$argmax_{y_1,..,y_T}\mathbb{P}(y_1, \ldots, y_T|x_1, \ldots, x_T) = argmax_{y_1,..,y_T}\frac{\mathbb{P}(x_1, \ldots, x_T|y_1, \ldots, y_T)\mathbb{P}(y_1, \ldots, y_T)}{\mathbb{P}(x_1, \ldots, x_T)}$$

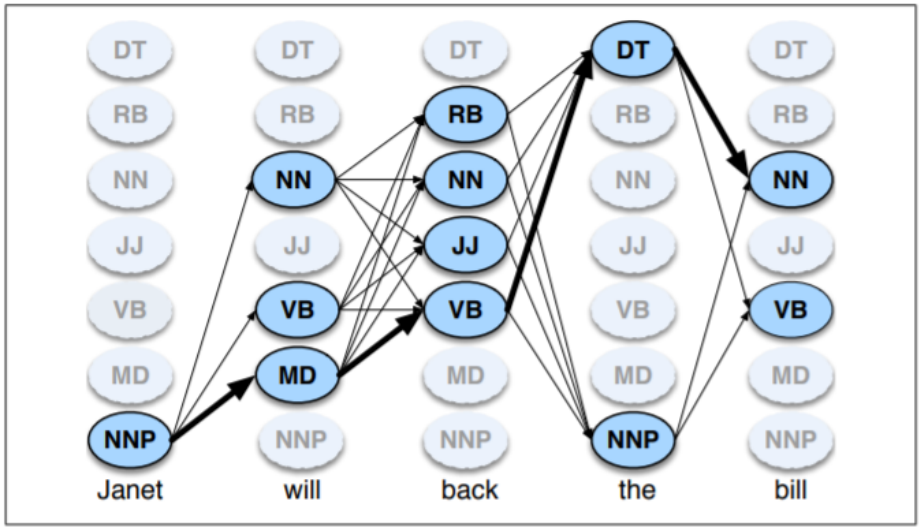
Since the word sequence $x_1, \ldots, x_T$ is given to us, so the above problem is equivalent to

$$argmax_{y_1,\ldots,y_T} \mathbb{P}(x_1, \ldots, x_T | y_1, \ldots, y_T) \mathbb{P}(y_1, \ldots, y_T)$$

We have our assumption that the unobserved states are from a Markov Chain and thus $\mathbb{P}(y_1, \ldots, y_T) = \prod_{i=1}^{T} \mathbb{P}(y_i | y_{i-1})$ and for $\mathbb{P}(y_1 | y_0)$ we use our initial distribution.
Since this is a **HMM**, so we also have $\mathbb{P}(x_1, \ldots, x_T | y_1, \ldots, y_T) = \prod_{t=1}^{T} \mathbb{P}(x_i | y_i)$
Thus we have reduced our problem to

$$argmax_{y_1,\ldots,y_T} \prod_{t=1}^{T} \mathbb{P}(x_i | y_i) \prod_{t=1}^{T} \mathbb{P}(y_i | y_{i-1})$$

now finding the tags $y_1, \ldots, y_T$ is not very easy. We demonstrate this with an example. Consider a sample sentence from a corpus. **"Janet will back the bill"**. Then there may be many tag sequences as seen below



Thus each such sequence gives rise to a path in the graph, and as such for the above figure there will be 7^5 many tags, out of which we need to find the tag which maximises the probability function. In general for N tags and T many words computing the probabilities of all the paths will be $O(N^T)$. To reduce the time complexity, we use the **Viterbi** algorithm which is a type of Dynamic Programming that overcomes the time complexity by storing the values of the previous step in a matrix. We successfully overcome the time complexity barrier, but not the space complexity barrier, since storing the matrix requires some (in some cases, a lot) memory. We discuss the Viterbi Algorithm in the next section.

3.2 Viterbi Algorithm

Consider a HMM model with given parameters. For an observed word sequence, we want to find the best sequence of tags i.e. we want to find $\max_{\mathbf{y}} \mathbb{P}(\mathbf{x}, \mathbf{y} | \lambda)$ or $\max_{\mathbf{y}} \mathbb{P}(\mathbf{y} | \mathbf{x}, \lambda)$. Now, we have N^T many possible choices of $\mathbf{y}$. Hence, we shall look for some ways to reduce the time complexity.

We define-

- $\delta_t(j) = \max_{y_{1:(t-1)}} \mathbb{P}(y_{1:(t-1)}, y_t = s_j | \mathbf{x}, \lambda)$ which the probability of observing the first t most likely tags ending with s_j.
- $\psi_t(i) = \hat{y}_{j-1}$ if $\hat{y_1}, \hat{y_2}, \cdots, \hat{y_{t-1}}, s_i$ is the most likely first t tag sequence ending with s_i.

We observe that $\delta_{t+1}(j) = \max_i \{\delta_t(i) a_{ij}\} b_{s_j}(x_{t+1})$. This observation is the heart of Viterbi algorithm. We will now use dynamic programming to solve our problem. We will store $\delta_t(i)$ and $\psi_t(i)$ values in two $N \times T$ arrays.

The Viterbi algorithm is as follows-

1. Initialize $\delta_1(i) = \pi_i b_i(y_1)$ and $\psi_t(i) = 0$.

2. Inductive step
    - $\delta_{t+1}(j) = \max_i\{\delta_t(i)a_{ij}\}b_{s_j}(x_{t+1})$
    - $\psi_{t+1}(j) = \text{argmax}_{1\leq i\leq N}\delta_t a_{ij}$
3. Termination $P^* = \max_{1\leq i\leq N}\{\delta_T(i)\}$, $Q^* = \text{argmax}_{1\leq i\leq N}\{\delta_T(i)\}$
4. Return the required label/tag sequence in reverse direction using the relation $\hat{y_T} = Q*$ and $\hat{y}_{t-1} = \psi_t(\hat{y}_t)$.

3.3 Forward Backward Algorithm

Forward-Backward algorithm gives a method to calculate the probability $\mathbb{P}(\mathbf{x}|\lambda)$. Using the usual brute-force method this probability can be written as-

$$\mathbb{P}(\mathbf{x}|\lambda) = \sum_{\mathbf{y}} \mathbb{P}(\mathbf{x},\mathbf{y}|\lambda) = \sum_{\mathbf{y}} \pi_{y_1} b_{y_1}(x_1) a_{y_1 y_2} b_{y_2}(x_2) \cdots a_{y_{T-1} y_T} b_{y_T}(x_T)$$

We observe that the above summation contains N^T terms which grows exponentially hence this summation is intractable by brute force approach. However the time complexity can be reduced if we use dynamic programming. We illustrate the idea in the following algorithms.

Forward algorithm-
Define $\alpha_t(i) = \mathbb{P}(x_{1:t}, Y_t = s_i|\lambda)$ which is defined as forward probability.
Algorithm

- $\alpha_1(i) = \pi_{s_i} b_{s_i}(x_1)$
- $\alpha_{t+1}(j) = \sum_{i=1}^N \alpha_t(i) a_{ij} b_{s_j}(x_{t+1})$ for $1 \leq t \leq T-1$
- Finally, $\mathbb{P}(\mathbf{x}|\lambda) = \sum_{i=1}^N \alpha_T(i)$

Backward Algorithm
Define $\beta_t(i) = \mathbb{P}(x_{t+1:T}|Y_t = s_i, \lambda)$ which is defined as backward probability.
Algorithm

- We define $\beta_T(i) = 1$
- $\beta_t(i) = \sum_{j=1}^N a_{ij}\beta_{t+1}(j) b_{s_j}(x_{t+1})$ for $1 \leq t \leq T-1$
- $\beta_0(i) = \sum_{j=1}^N a_{ij}\beta_{t+1}(j) b_{s_j}(x_{t+1})$ for $1 \leq t \leq T-1$
- Finally, $\mathbb{P}(\mathbf{x}|\lambda) = \sum_{i=1}^N \beta_0(i)$

We observe that we can calculate the required probability using Forward algorithm but we calculated the same probability using Backward algorithm because we need both $\alpha_t(i)$ and $\beta_t(i)$ to find the maximum likelihood estimate of the model parameters using the EM Algorithm/ Baum-Welch Algorithm.

3.4 Learning Model Parameters

3.4.1 Baum-Welch Algorithm

Until now we assumed that we know the model parameters beforehand. We shall now consider the problem of finding λ_{mle} given the observations $\mathbf{x}$ we want to find $\lambda_{mle} = \text{argmax}_\lambda \mathbb{P}(\mathbf{x}|\lambda)$. Before proceeding to the algorithm we make a few definitions-

Definition

- $\xi_t(i,j) = \mathbb{P}(Y_t = s_i, Y_{t+1} = s_j|\mathbf{x},\lambda)$ where $1 \leq t \leq T-1$
- $\gamma_t(i) = \mathbb{P}(Y_t = s_i|\mathbf{x},\lambda) = \sum_{j=1}^N \xi_t(i,j)$ where $1 \leq t \leq T-1$

Now, $\xi_t(i,j)$ can be given by-

$$\begin{aligned}\xi_t(i,j) &= \mathbb{P}(y_t = s_i, q_{t+1} = s_j|\mathbf{x},\lambda) \\ &= \frac{\alpha_t(i)a_{ij}b_j(x_{t+1})\beta_{t+1}(j)}{p(\mathbf{x}|\lambda)} \\ &= \frac{\alpha_t(i)a_{ij}b_j(x_{t+1})\beta_{t+1}(j)}{\sum_i\sum_j \alpha_t(i)a_{ij}b_j(x_{t+1})\beta_{t+1}(j)}\end{aligned}$$

Now,

- $\sum_{t=1}^{T-1}\xi_t(i,j)$ is the expected no of transitions from s_i to s_j given the observed values $\mathbf{x}$ and parameter λ.
- $\sum_{t=1}^{T-1}\gamma_t(i)$ is the expected no of transitions from s_i to some state given the observed values $\mathbf{x}$ and parameter λ.

Hence, given $\mathbf{x},\lambda$ we can calculate $\alpha_t(i),\beta_t(i),\xi_t(i,j),\gamma_t(i)$. We now discuss the Baum Welch algorithm which can be used to find the optimal λ given the observed values $\mathbf{x}$

Baum Welch algorithm

- Initialise $\lambda^{(0)}$
- For a given λ^k we calculate $\alpha_t^{(k)}(i),\beta_t^{(k)}(i),\xi_t^{(k)}(i,j),\gamma_t^{(k)}(i)$
- Update $\lambda^{k+1} = (A^{k+1},B^{k+1},\pi^{k+1})$ as

$$\begin{aligned}\pi_i^{k+1} &= \gamma_1^{(k)}(i) \\ a_{ij}^{k+1} &= \frac{\sum_{t=1}^{T-1}\xi_t^{(k)}(i,j)}{\sum_{t=1}^{T-1}\gamma_t^{(k)}(i)} \\ b_{ij}^{k+1} &= \frac{\sum_{t=1}^{T-1}1(x_t=w_j)\gamma_t^{(k)}(i)}{\sum_{t=1}^{T-1}\gamma_t^{(k)}(i)}\end{aligned}$$

- Continue step 2,3 until convergence.

If we observe several sequences $\mathbf{y}_1,\mathbf{y}_2,\cdots,\mathbf{y}_R$ where $\mathbf{y}_r = (y_{1r},\cdots,y_{N_r r})$, then we can update our model parameters as-

$$\begin{aligned}\pi_i^{k+1} &= \frac{\sum_{r=1}^{R}\gamma_1^{(k)}(i,r)}{R} \\ a_{ij}^{k+1} &= \frac{\sum_{r=1}^{R}\sum_{t=1}^{T-1}\xi_t^{(k)}(i,j,r)}{\sum_{r=1}^{R}\sum_{t=1}^{T-1}\gamma_t^{(k)}(i,r)} \\ b_{ij}^{k+1} &= \frac{\sum_{r=1}^{R}\sum_{t=1}^{T-1}1(x_t=w_j)\gamma_t^{(k)}(i,r)}{\sum_{r=1}^{R}\sum_{t=1}^{T-1}\gamma_t^{(k)}(i,r)}\end{aligned}$$

We observe that Baum-Welch Algorithm provides an unsupervised learning method.

3.4.2 Maximum Likelihood Estimate

Now, we shall now assume that we know the hidden states i.e. our the words of the sentences are tagged and we want to find $\lambda_{mle} = \max_\lambda p(\mathbf{x},\mathbf{y}|\lambda)$

For observed sequences $(\mathbf{x}_1,\mathbf{y}_1),\cdots,(\mathbf{x}_R,\mathbf{y}_R)$ of observed states and hidden states, we define-

$$
\begin{aligned}
N_j^1 &= \sum_{r=1}^{R} \mathbf{1}(y_{1r} = j) = \#\text{No of times, the hidden state sequence starts from state } j \\
N_{jk} &= \sum_{r=1}^{R} \sum_{t=1}^{T_r - 1} \mathbf{1}(y_{t,r} = j, y_{(t+1),r=k}) = \#\text{No of times, the hidden state jumps from state } j \text{ to state } k \\
N_{jl}^X &= \sum_{r=1}^{R} \sum_{t=1}^{T_r} \mathbf{1}(y_{tr} = j, x_{tr} = l) = \#\text{No of times, the hidden state } j \text{ gives observed state } j \\
N_j &= \sum_{l=1}^{M} N_{jl} = \#\text{No of times, the hidden state is } j
\end{aligned}
$$

The required maximum likelihood estimates are-

$$
\begin{aligned}
\hat{\pi} &= \frac{N_j^1}{\sum_j N_j^1} \\
\hat{a}_{jk} &= \frac{N_{jk}}{\sum_k N_{jk}} \\
\hat{b}_{jl} &= \frac{N_{jl}}{N_j}
\end{aligned}
$$

The calculations of MLE has been shown in [5]

4 Data Analysis

We used the **treebank** data available in the Python **nltk** library. There are **3914** tagged sentences which comprise of **100676** tagged words. Note that there are 46 distinct tags in the dataset. We considered the **training data** to be the first 80% of tagged sentences. We chose such a division because we wanted to include as much data for training as possible, and also have enough testing data so that we could have labels from most of the categories in the test data.

4.1 Results for Classification based approach

We performed the feature extraction for the i-th word as follows:

- word preceding the i-th word
- word succeeding the i-th word
- i-th word itself
- 1 letter prefix of i-th word
- 2 letter prefix of i-th word
- 3 letter prefix of i-th word
- 1 letter suffix of i-th word
- 2 letter suffix of i-th word
- 3 letter suffix of i-th word
- 1 if it is first letter of sentence, 0 else.
- 1 if it is last letter of sentence, 0 else.
- 1 if i-th word is all capitalized, 0 else.
- 1 if i-th word is all lower case, 0 else.

- 1 if i-th word consists of only alphanumeric characters, 0 else.

These features capture a lot of important information in the setup. Now we will use this feature representation to train our classifiers on the **training data** which we took to be the first 80% of tagged sentences, and we use the remaining 20% data as the test data. We chose such a split because we wanted to have enough sentences for training, and at the same time enough sentences in the test so that it encompasses all the 46 tags of the treebank dataset.

After one hot encoding the features, we observe that the resulting feature matrix X_train is a sparse matrix of size 80637×42386. We observe that it is a very high dimensional data matrix ! We shall try various classification methods on this data.

Note that we reported the precision, recall and F1 score averaged over all the groups instead of the usual weighted average. This is because we saw that some of the common , **large** POS tag-classes occured very frequently and the classifier performed quite well. However, the classifier did some misclassification on words originally from rare , **small** pos tag-classes.

4.1.1 Logistic Regression Classifier

We trained a logistic regression based classifier on the training data using default 'lbfgs' solver. We **did not achieve convergence** , but the classifier manages to give average macro precision of 0.93, average macro recall and F1 score of 0.90 each.

Non convergence of algorithm might be due to the fact that there is multi-collinearity in the data or that the data might be perfectly linearly separable

4.1.2 Bernoulli Naive Bayes Classifier

Our feature matrix is such that each of it's rows has binary $0/1$ entries. Hence, we can assume that they are from some Bernoulli distribution. Under the added Naive Bayes assumption we trained a Bernoulli Naive Bayes Classifier. We observe that it has $< 50\%$ recall and F1 score. Hence, it is a bad candidate for the classifier.

4.1.3 K-nearest Neighbour Classifier

We fitted a knn classifier for $k = 3, 5$ getting almost identical results in each case with an average macro accuracy of 0.88% precision, recall and F1 score.

4.1.4 SVM Classifier

The SVM Classifier was successfully trained on our training data and then we observed the best results with this approach (as compared to all the classification methods.) with macro average accuracy of 92% precision, recall and F1 score. Also, the weighted average accuracy is 96% precision, recall and F1 score. We had taken the C parameter equal to 1.

Note that we did not move around the C values a lot because choosing a high value or low value of C will essentially depend on the nature of future data that the classifier will encounter . This is quite hard to say in a higher dimension situation like this.

4.1.5 Some observations

- We did not apply Classification Tree because the fact that we are dealing with such high dimensional data makes it prone to overfitting. Also, we are not very sure about whether the type of 'rectangular' partioning of the sample space is a good thing to do.
- We could not apply classification algorithm like Fisher's LDA because we couldn't find Python libraries that performed LDA on a sparse matrix . We tried to load the matrix in memory, but the Colab notebook crashed every time.
- One hot encoding drastically increases the dimension of our feature space and it leads to a host of computational problems. So we could have tried some 'word embeddings' that reduce the dimension by representing all words in a space of smaller dimension (say 50 or 100). There is still an issue of what to do with out of vocab (**oov**) words during the training phase and testing phase.
- There are pre-trained language models like **BERT** that provide contextual embeddings for a huge vocabulary of words, but unfortunately they are deep inside the realm of deep-learning, and hence we chose not to venture into that space.

4.2 Results for sequence to sequence based approach

We achieved 0.84 macro average precision, 0.86 average macro recall and 0.82 macro average f1 score.

It roughly performed better on an average on larger, more common class, and made more errors on the less common tags.

4.2.1 Some observations

- We believe that the HMM based tagging model needs a lot more data compared to the linear classifiers. This is because we are estimating a lot more parameters $(\#\text{tags})^2 + (\#\text{tags}) \times (\#$ distinct words in training corpus$)$. We need more data to train our HMM model so that it may give better performance.
- HMM model essentially disregards sub-word level features that we could extract and use in our classification based approach.
- Incorporating unknown words is also a challenge in this case.

References

[1] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[2] James H. Martin Dan Jurafsky. " Speech and Language Processing". In: (). URL: https://web.stanford.edu/~jurafsky/slp3/.

[3] Merriam-Webster.com Dictionary. *part of speech*. URL: https://www.merriam-webster.com/dictionary/part%5C%20of%5C%20speech. (accessed January 31, 2022).

[4] Jesús Giménez and Lluis Marquez. "Fast and accurate part-of-speech tagging: The SVM approach revisited". In: *Recent Advances in Natural Language Processing III* (2004), pp. 153–162.

[5] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series)*. The MIT Press, 2012.

[6] Tetsuji Nakagawa, Taku Kudo, and Yuji Matsumoto. "Unknown Word Guessing and Part-of-Speech Tagging Using Support Vector Machines." In: *NLPRS*. Citeseer. 2001, pp. 325–331.