**NATURAL LANGUAGE PROCESSING CS6030**

***PoS Tagger in NLTK***

Date : 27-10-2021

*Praveen RS*

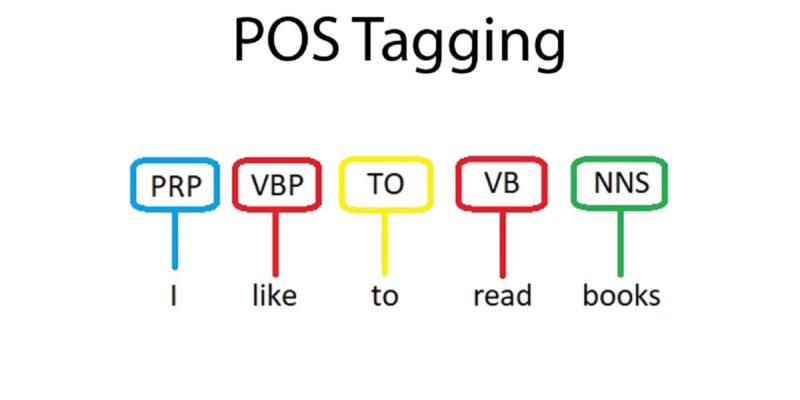
*2018103577*

*P Batch*

**Problem Statement**

This assignment focuses on building a POS Tagger with a Bidirectional LSTM model and train it using a supervised dataset of sentences with each word tagged with their respective Part of Speech. **Part-of-speech (POS)** tagging is a popular Natural Language Processing process which refers to categorizing words in a text (corpus) in correspondence with a particular part of speech, depending on the definition of the word and its context. From the model developed, we can get the POS outputs for custom inputs(sentences).

Part-of-Speech (PoS) tagging, then it may be defined as the process of assigning one of the parts of speech to the given word. It is generally called POS tagging. In simple words, we can say that POS tagging is a task of labelling each word in a sentence with its appropriate part of speech. We already know that parts of speech include nouns, verb, adverbs, adjectives, pronouns, conjunction and their sub-categories.



Most of the POS tagging falls under Rule Base POS tagging, Stochastic POS tagging and Transformation based tagging.

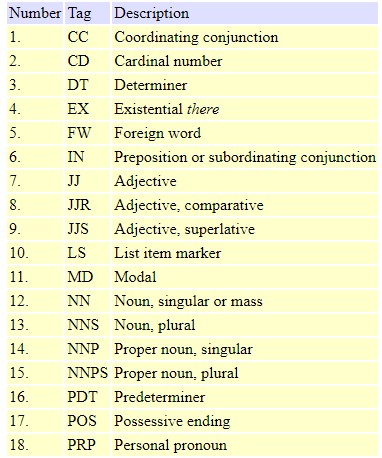
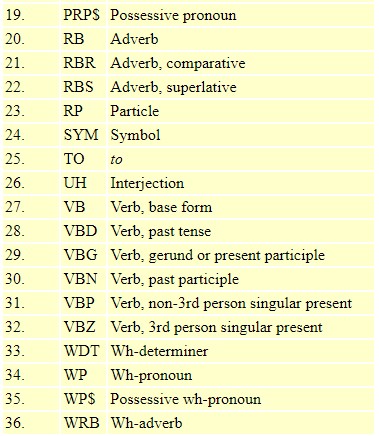
**Dataset details**

We can apply machine learning models and rule-based models to obtain the parts of speech tags of a word. The most commonly used parts of speech tag notations are provided by the **Penn Treebank corpus**. In which, a total of 48 P.O.S tags are defined according to their usage.

The **English Penn Treebank tagset** is used with English corpora annotated by the TreeTagger tool, developed by Helmut Schmid in the TC project at the Institute for Computational Linguistics of the University of Stuttgart. This version of the tagset contains modifications developed by Sketch Engine (earlier version).

We will be using this tagset for training the model so that it will able to perform POS tagging for our custom inputs.

Alphabetical list of part-of-speech tags used in the Penn Treebank Project:

** **

**Modules Used**

**1.Loading the dataset and necessary libraries**

We’ll be building a POS tagger using Keras and a Bidirectional LSTM Layer. Firstly, all the necessary libraries are imported and we use a corpus that’s included in NLTK.

**2. Dataset Pre-processing**

Before training a model, we need to split the data in training and testing data. As usual, let’s use the train\_test\_split function from Scikit-Learn after separating the tags from the words in the sentences.  We’re computing a set of unique words (and tags) then transforming it in a list and indexing them in a dictionary. These dictionaries are the word vocabulary and the tag vocabulary. Next, we’ll convert these word and tag datasets to integer datasets for computations. Hence, all the words and tags are encoded into integers.

**3. Word Embedding**

**Word Embedding** is one of the popular representations of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc. Basically, it's a feature vector representation of words which are used for other natural language processing applications.

We should feed in a sequence of numbers to it and also we should ensure there is no variance in input shapes of sequences. It all should be of same length. But texts in sentences have different count of words in it. To avoid this, we seek a little help from pad\_sequence to do our job before word embedding. It will make all the sequence in one constant length, which in our case will be the maximum length of all sentences.

**4. Model Creation and Training**

We are clear to build our Deep Learning model. While developing a DL model, we should keep in mind of key things like Model Architecture, Hyperparmeter Tuning and Performance of the model.

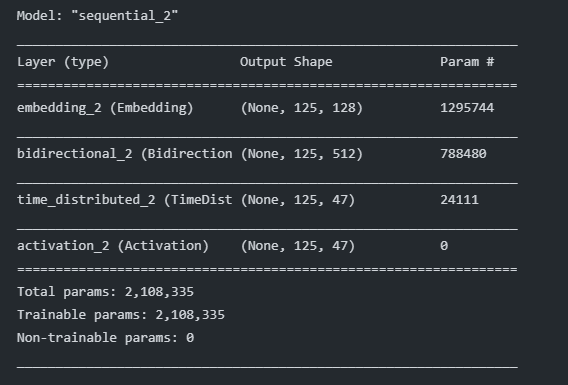
For model architecture, we use:

**1) Embedding Layer** - Generates Embedding Vector for each input sequence.

**2) Bidirectional LSTM** - **Bidirectional modifier** inputs to the **LSTM** the next values in the sequence, not just the previous. This layer is mainly used in such sequence processing problems. We need to set the **return\_sequences** parameter as True so that the LSTM outputs a sequence, not only the final value.

**3) Time Distributed Layer** – Since the dense layer needs to run on each element of the sequence, we need to add the TimeDistributed modifier.

* **4) Activation Layer** - Softmax is used as the activation function because this is a multi-class classification problems where class membership is required on more than two class labels.



For training the model, we have to set appropriate values to a few parameters:

|  |  |
| --- | --- |
| Optimizer | Adam |
| Loss | Categorical Cross-entropy |
| Epochs | 40 |
| Batch size | 128 |
| Learning rate | 0.001 |
| Metrics | Ignore Class Accuracy |

We have used a custom accuracy as the metric because for most of the sentences, the largest part is “padding tokens”. These are really easy to guess, hence the super high performance. Hence, we use this accuracy metric that ingores the paddings.

**5.Result Analysis**

After the model is trained, it is tested against some custom sentences to give accurate outputs in the form of POS tags.

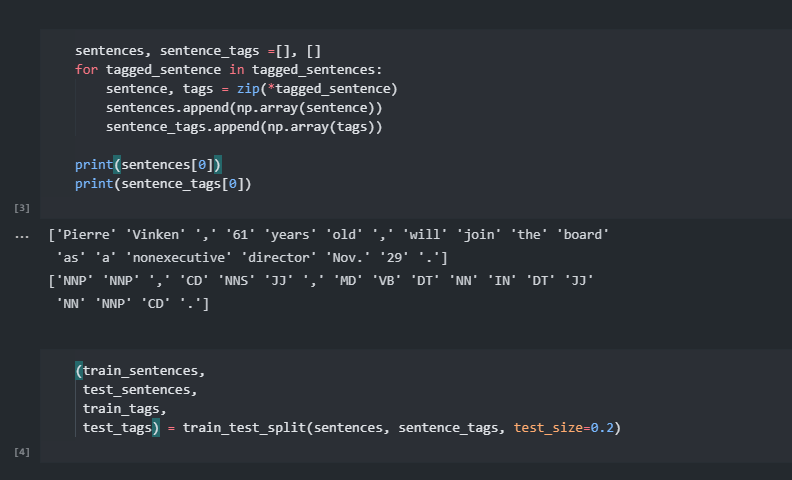
**Source Code**

*Importing the necessary libraries and datasets*

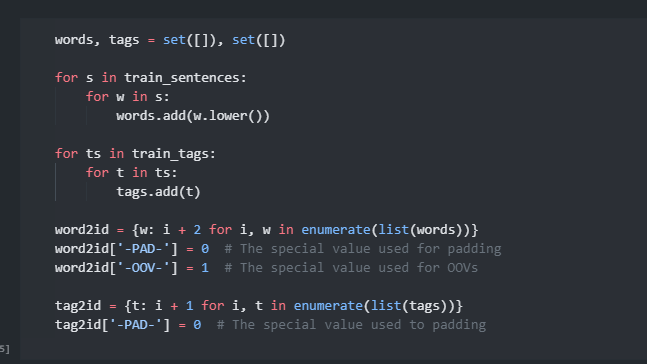
*A sample sentence and its corresponding POS tags are displayed*

****

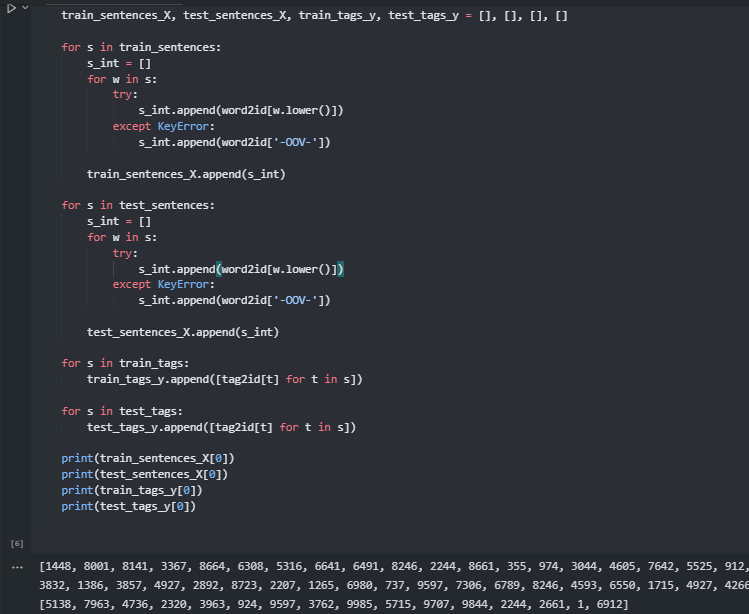
*The tags and words are separated and then into train and test datasets*

****

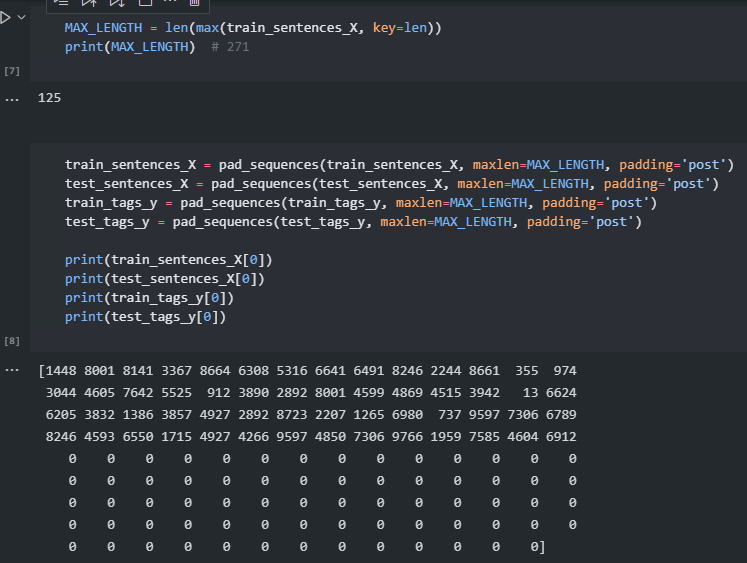
*Words and tags are made into dictionaries for encoding into numbers. Special tags ‘-PAD-’ for padding and ‘-OOV-’ for out of vocabulary words are used*

****

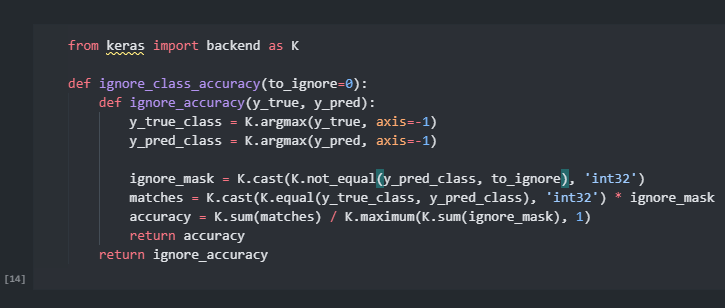
*Using the word2id and tag2id dictionaries to encode them into integers, i.e. their indices from the dictionaries.*

****

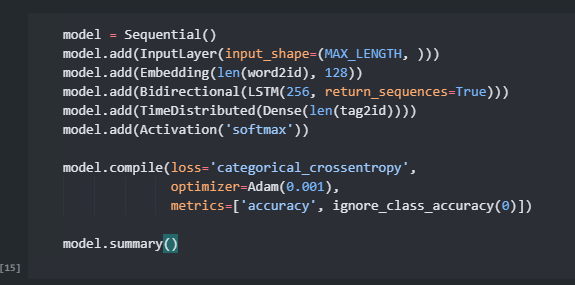
*Padding the sentences with 0’s at the end*

****

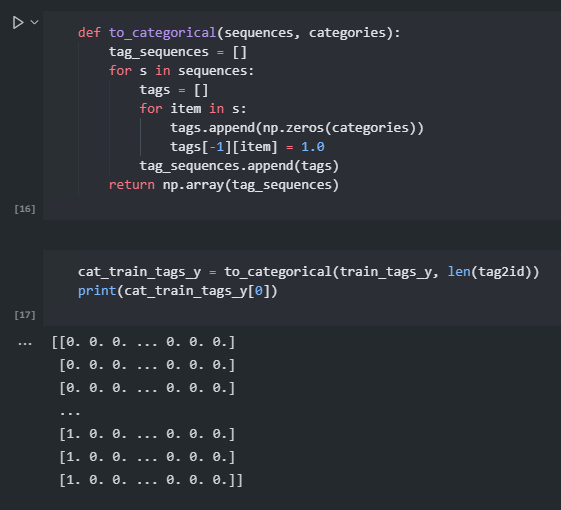
*Custom accuracy metric defined for better results*

****

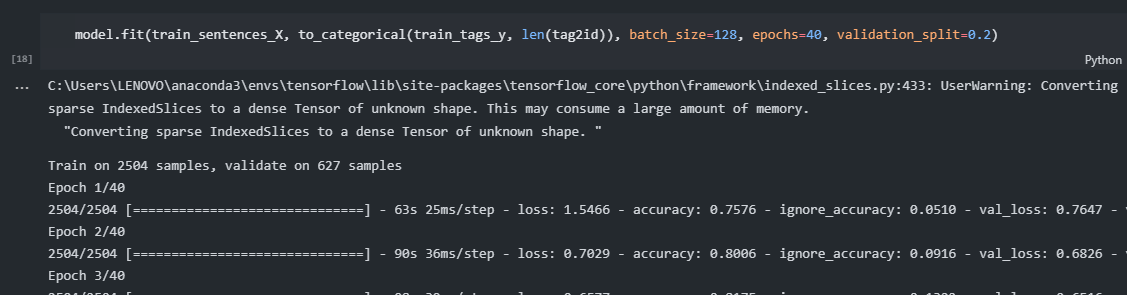
*Building the model*

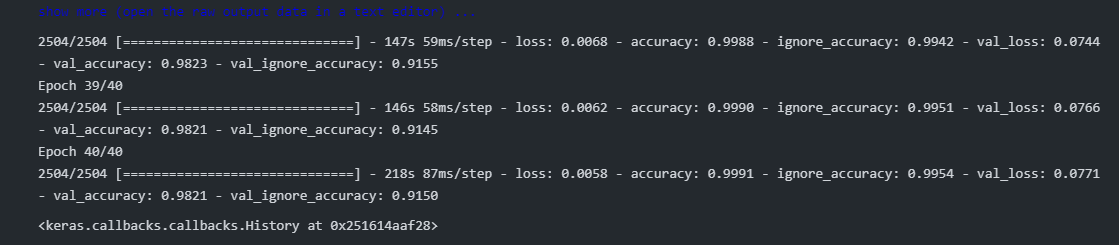
****

*There’s one more thing to do before training. We need to transform the sequences of tags to sequences of One-Hot Encoded tags.*

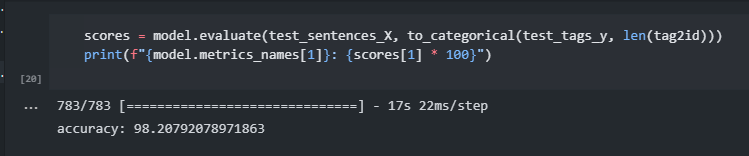
****

*Training the model for 40 epochs*

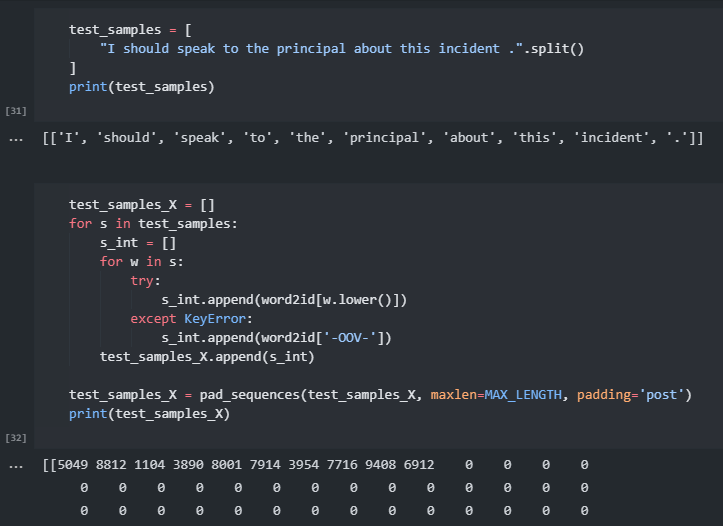
**

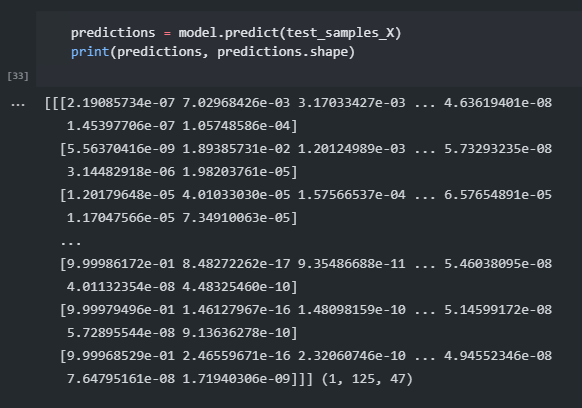
**

*Accuracy on predicting the test data is around 98%*

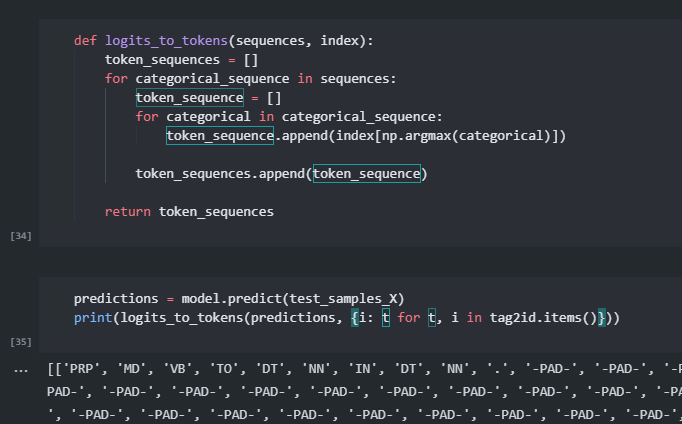
****

*Testing on custom input*

**

**

*We need to do the “reverse” operation for to\_categorical to get out outputs back in the form of Tags instead of integers*



*Hence, we get the correct POS tagging for the sentence: “I should speak to the principal about this incident.”*