POS TAGGING

Natural Language Processing

Syed Muhammad Zuhair Abbas

20.10.2020

B17 - DS - 02

>> **COLLAB LINK** <<

INTRODUCTION

Parts of speech tagging have a paramount importance in Natural language processing. It is used to classify the words into respective parts of speech or lexical categories. POS tagging has wide applications including sentiment analysis, question answering, word sense disambiguation, etc. In this project, we used a supervised learning approach to implement a tagger in Python. We will explain our method in the following sections.

DATASET

We used an ancient Indo-Aryan language, **Vedic Sanskrit**. The language is of historical value in South Asia. Using the tagging program, lexical and morpho-syntactic information was created and manually validated. POS tags are automatically induced from the morpho-syntactic data of each word. The dataset contains **2524 training** and **1473 test** sentences.

FEATURES GENERATION

The dataset only provides us with the words and their respective tags. In order to build a strong classifier we need to add more features. In our case we created a simple yet very insightful feature set containing the information such as positioning, information about capitalization of first letter, previous and next words, as well as prefixes and suffixes. We map our sentence's list to a list of dictionaries containing the respective features.

FEATURES ENCODING

Before passing the features as input to the model, we need to encode them into vectors. We used a builtin function, **DictVectorizer** provided by **Scikit learn** for this purpose. The DictVectorizer was fit on both train and test data, thus it automatically handled the missing words.

LABELS ENCODING

Similarly, we also need to encode our labels. We used a **label encoder** which maps the Strings to digits. Our output variable contained **11** different values encoded as integers.

CLASSIFICATION

We trained 2 different classifiers and compared their performances:

- **DECISION TREE**

  Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data feature. We used sk-learn to train the decision tree classifier.

- **DEEP NEURAL NETWORK**

  Deep Neural Networks (DNNs) are typically Feed Forward Networks (FFNNs) in which data flows from the input layer to the output layer without going backward. Our model had 4 dense layers and dropout layer in the middle to avoid overfitting .The following hyper-parameters were fine tuned using grid-search:

  - Optimizer : NAdam, Learning rate: 0.0005, Epoch: 7, Dropout: 0.2
  - Loss: Sparse Categorical Cross Entropy, Batch size: 128.

RESULTS

Below are the prediction accuracies obtained by our models on the test dataset. This division of data is only done on Training set, the testing data remains the same.

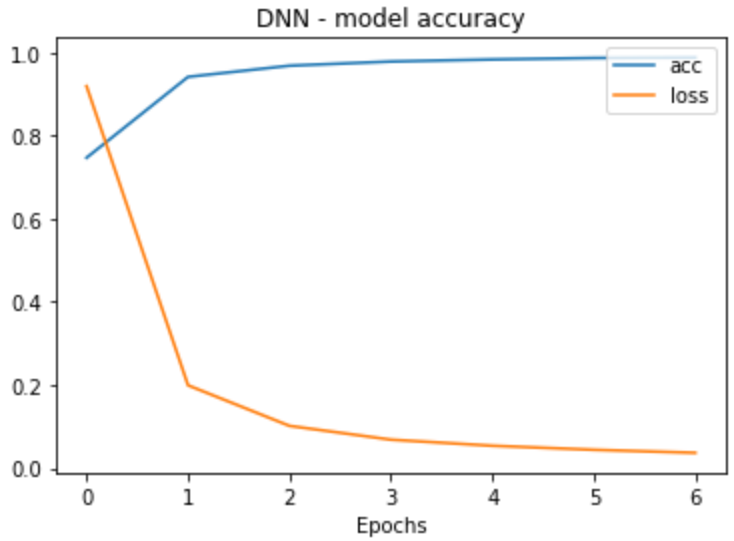
TRAINING DATA %	DECISION TREE ACC %	NEURAL NETWORK ACC %
10	76.42	78.26
20	84.70	85.17
30	87.01	87.46
40	88.65	88.66
50	88.90	90.20
60	89.49	90.39

70	90.18	91.25
80	90.18	91.54
90	91.08	92.18
100	**91.66**	**92.89**

The trend shows that the more training data we have the better our classifier performs on test data.

ANALYSIS

We can observe that both classifiers managed to obtain high accuracy and precision but the Neural network still outperformed a Decision tree classifier. The training was really time efficient and smooth and we didn't notice any bias or variance.



We had enough data to train however not all the classes were equally balanced eg; Noun & verb class collectively covered more than 50% of data in both training and test. This is the main reason that most words were either misclassified as nouns or verbs. Below are the number of elements belonging to each class in the training and testing dataset as well as the number of misclassifications for each of the Parts of speech respectively:

```
Total lables Train: 17445

Counter({'ADJ': 1503,
         'ADV': 1860,
         'AUX': 211,
         'CCONJ': 462,
         'DET': 117,
         'NOUN': 5768,
         'NUM': 203,
         'PART': 1105,
         'PRON': 2705,
         'SCONJ': 168,
         'VERB': 3343})
```

```
Total lables Test: 9672

Counter({'ADJ': 870,
         'ADV': 1084,
         'AUX': 90,
         'CCONJ': 152,
         'DET': 48,
         'NOUN': 3074,
         'NUM': 89,
         'PART': 785,
         'PRON': 1443,
         'SCONJ': 97,
         'VERB': 1940})
```

```
Incorrect Predictions on Test data:

NOUN     367
VERB     130
ADJ      108
ADV       74
PRON      72
DET       16
AUX       15
NUM       12
SCONJ      8
PART       4
Name: predicted, dtype: int64
```

Considering the unbalanced class problem, the tagger still managed to provide really good results. There were few errors which also occured due to the context specific use of a word which can be classified as more than one part of speech depending on the use. Following are a few examples of correctly and incorrectly classified words respectively:

	input	actual	predicted
0	yad	PRON	PRON
3	viśva	DET	DET
4	rūpa	NOUN	NOUN
5	bhṛ	VERB	VERB
6	vācaspati	NOUN	NOUN

	input	actual	predicted
1	triṣapta	ADJ	NOUN
2	parī	VERB	NOUN
16	deva	ADJ	NOUN
53	śru	VERB	NOUN
58	śru	VERB	NOUN

CONCLUSION

In this project, we learn how to implement our own POS tagger. We trained 2 different classifiers and compared their performances. We analyzed how we can improve our feature set and how to tackle the problems we might face in the implementation of language processing tasks.

REFERENCES

- Universal Dependencies
- UD_Sanskrit-Vedic
- POS tagging applications
- Decision Tree
- DNN