

**International Institute of Informational Technology,
Hyderabad**

Project on

**Implementing SVM based POS tagger on
Hindi, Telugu and Bengali ILCI datasets
and finding which set of features
performed the best**

*Under the guidance of
Faculty: Prof. Dipti M.Sharma
Mentor : Prathiba Rani*

*by :
Rajkumar Kandala
Mahesh Natakari
IIIT, Basar*

Introduction

In Natural Language Processing, in order to do any nlp operations like Text analysis, Information Retrieval and Extraction, Summarization, first the model has to do understand the text given to it. To achieve this, model has to understand meaning and identify the parts of speech tag (eg: noun, verb, adjective etc.) of each word i.e POS tagging.

The tool, which is used to do pos tagging for the given dataset based on support vector machines is called **SVM TOOL**.

Description of project

Our project is to Implement SVM based POS tagger on Hindi, Telugu and Bengali ILCI datasets and finding which set of features performed the best.

SVMTool

The SVMTool software package consists of three main components, namely the learner (SVMTlearn), the tagger (SVMTagger) and the evaluator (SVMTeval), which are described below.

1.SVMTlearn :

it is responsible for the training set of examples (either annotated or unannotated) of SVM classifiers.

2.SVMTagger :

it performs the POS tagging of a sequence of words.

3.SVMTeval :

SVMT eval evaluates the performance in terms of accuracy.

Sample Feature set

word features	$w_{-3}, w_{-2}, w_{-1}, w_0, w_{+1}, w_{+2}, w_{+3}$
POS features	$p_{-3}, p_{-2}, p_{-1}, p_0, p_{+1}, p_{+2}, p_{+3}$
ambiguity classes	a_0, a_1, a_2, a_3
may_be's	m_0, m_1, m_2, m_3
word bigrams	$(w_{-2}, w_{-1}), (w_{-1}, w_{+1}), (w_{-1}, w_0),$ $(w_0, w_{+1}), (w_{+1}, w_{+2})$
POS bigrams	$(p_{-2}, p_{-1}), (p_{-1}, a_{+1}), (a_{+1}, a_{+2})$
word trigrams	$(w_{-2}, w_{-1}, w_0), (w_{-2}, w_{-1}, w_{+1}),$ $(w_{-1}, w_0, w_{+1}), (w_{-1}, w_{+1}, w_{+2}),$ (w_0, w_{+1}, w_{+2})
POS trigrams	$(p_{-2}, p_{-1}, a_{+0}), (p_{-2}, p_{-1}, a_{+1}),$ $(p_{-1}, a_0, a_{+1}), (p_{-1}, a_{+1}, a_{+2})$
sentence_info	punctuation ('.', '?', '!')
prefixes	$s_1, s_1 s_2, s_1 s_2 s_3, s_1 s_2 s_3 s_4$
suffixes	$s_n, s_{n-1} s_n, s_{n-2} s_{n-1} s_n, s_{n-3} s_{n-2} s_{n-1} s_n$
binary word features	initial Upper Case, all Upper Case, no initial Capital Letter(s), all Lower Case, contains a (period / number / hyphen ...)
word length	integer

Implementation

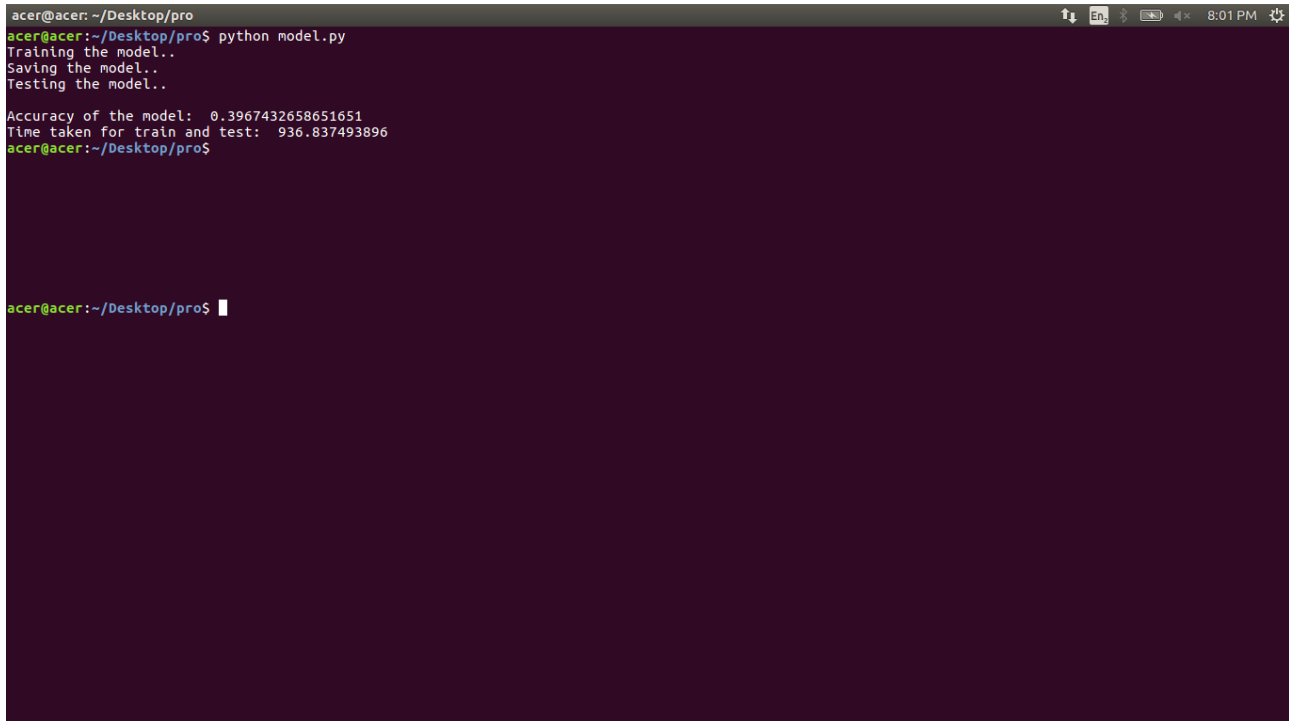
- i. In the first step, we have converted the data set which is in the ssf format into list of sentences in which a sentence is again a list of words.
- ii. After that we created word vectors from the sentences which we extracted from the dataset , using word2vec model by giving vector dimension is 5, window size is 3 and minimum word count is 1 as parameters.
- iii. These vectors are given as input to python SVM classifier which is inbuilt in sklearn module.
- iv. We trained and tested the model with 10% of training data with set of features which resulted in accuracy of 61%.

Sample output :

```
acer@acer: ~/Desktop/project
acer@acer:~/Desktop/project$ python model_bi\&trigrams.py
weighted f1_score of the model: 0.6127903119250955
acer@acer:~/Desktop/project$
```

- v. Another set of features we used are word itself, bigrams, trigrams and word length. Using these features the model had given an accuracy of 39% with 10% training data.

Sample output :

A screenshot of a terminal window with a dark purple background. The window title bar shows 'acer@acer: ~/Desktop/pro' and system icons on the right including a volume icon, a network icon, a battery icon, and the time '8:01 PM'. The terminal text shows the execution of 'python model.py', followed by status messages: 'Training the model..', 'Saving the model..', and 'Testing the model..'. The results displayed are 'Accuracy of the model: 0.3967432658651651' and 'Time taken for train and test: 936.837493896'. The prompt 'acer@acer:~/Desktop/pro\$' is shown at the bottom with a cursor.

```
acer@acer:~/Desktop/pro
acer@acer:~/Desktop/pro$ python model.py
Training the model..
Saving the model..
Testing the model..

Accuracy of the model: 0.3967432658651651
Time taken for train and test: 936.837493896
acer@acer:~/Desktop/pro$

acer@acer:~/Desktop/pro$
```

Conclusion:

We have obtained the good results till now. We are still working on this project to increase the efficiency and accuracy of the model.