CMPE561
Natural Language Processing Course
An HMM POS Tagger Implementation with Viterbi Decoding

Samet Atdag[1]

[1]*Department of Computer Engineering, Bogazici University, Istanbul*

(Dated: May 5, 2016)

In this homework, an HMM Part-of-Speech Tagger with Viterbi decoding is implemented to detect the POS tags of a given text. turkish_metu_sabanci_*.conll dataset is used for training and validation. The performance of the tagger is calculated using overall accuracy. A confusion matrix for different types of tags (cpostag, postag) is given.

The code of the project is in Github account: https://github.com/sametatdag/CMPE561.

Contents

I. INTRODUCTION

In corpus linguistics, part-of-speech tagging is the process of tagging a word according to it's duty in terms of its definition and its context. Context is usually defined as its relationship with adjacent and related words in a phrase, sentence or paragraph. Hidden Markov Models provides a large series of opportunities for detection of hidden relationships within a series of information. So that, HMM for POS tagging is used very commonly.

In this study, an HMM POS tagger with Viterbi Decoding is implemented to detect the POS tags of a given text. The purpose is building and training an HMM to predict the POS tags of the test input file with a certain performance.

II. DATASET

- turkish_metu_sabanci_train.conll is used for training. It contains 5000 manually labeled sentences.
- turkish_metu_sabanci_val.conll is used for validation. It contains 300 manually labeled sentences.

III. PRE-PROCESSING AND TOKENIZATION

Since a CONLL input files has a certain structure, no pre-processing was required.

A. Sentence Tokenization

In CONLL file, sentences are separated from each other with 2 consecutive newline characters. While reading the file, content is splitted by two newlines to tokenize the sentences.

B. POS Tag Type

Trainer script accepts the POS Tag type: cpostag or postag. CPOSTAG and POSTAG info in CONLL file lays in 3rd and 4th columns. Trainer script uses the pos tag type which is chosen with the parameter.

IV. HMM POS TAGGER WITH VITERBI DECODING

In this section, I try to clarify how parts of the project works to detect the POS tags. All the implementation is done from scratch; no ready libraries are used.

A. Training the HMM

For the model, tag transition probabilities (Fig. 1) and observation likelihoods (Fig. 2) are calculated using bi-gram word counts.

FIG. 1: Tag transition probabilities.
graphics[width=4cm]tagtransition

FIG. 2: Word likelihood probabilities.

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

Instead of training the HMM for each run, trained parameters are saved into 2 files when the training script is finished:

- tag_transitions.txt : Contains tag transition probabilities in JSON format.
- observation_likelihoods.txt : Contains word likelihoods in JSON format.

B. Tagging the novel data

Tagging is done via running hmm_tagger.py. This script splits the given text into sentences, decodes the input using a Viterbi backtracing matrix and creates an output file.

Created output file contains 3 column structure: WORD Estimated_Tag True_Tag

Sidenote:This structure is slightly different than what is defined in the homework text. Normally, True_Tag should be in a gold standard file. But since the given test file contains a very tiny dataset (only two sentences), we are advised to use the validation set for the testing purposes. So that, for the sake of convenience, I put gold standard into this output file.

C. Evaluating the results

To evaluate the accuracy, we should run evaluate_hmm_tagger.py. This file takes the output file which was generated at the end of previous step, calculates the accuracy and prints out.

Also, this script prints out a confusion matrix.

V. PERFORMANCE EVALUATION

In this section, the performance of the classifier is given. Overall accuracy for two different POS Tag types (cpostag, postag) is calculated for performance evaluation. Also confusion matrices for them is given.

A. Accuracy Results

	CPOSTAG	POSTAG	Average
Accuracy	27,011%	27,255%	27,133%

TABLE I: Overall Accuracy for CPOSTAG and POSTAG

	Adv	Punc	Noun	Adj	Verb	Det	Pron	Postp	Conj	Ques	Num	Interj	Dup	Zero
Adv	0	60	86	0	78	0	0	0	0	0	0	0	0	0
Punc	300	59	102	0	69	0	0	0	0	0	0	0	0	0
Noun	0	460	747	0	491	0	0	0	0	0	0	0	0	0
Adj	0	142	173	0	142	0	0	0	0	0	0	0	0	0
Verb	0	261	379	3	424	0	0	0	0	0	0	0	0	0
Det	0	38	71	0	94	0	0	0	0	0	0	0	0	0
Pron	0	22	29	0	27	0	0	0	0	0	0	0	0	0
Postp	0	24	35	0	20	0	0	0	0	0	0	0	0	0
Conj	0	31	58	0	43	0	0	0	0	0	0	0	0	0
Ques	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Num	0	8	23	0	11	0	0	0	0	0	0	0	0	0
Interj	0	0	0	0	2	0	0	0	0	0	0	0	0	0
Dup	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Zero	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TABLE II: Confusion matrix for POSTAG type. Rows represent true tags, columns represent estimated tags.

	Adv	Punc	Noun	Adj	Verb	Det	Pron	Postp	Conj	Ques	Num	Interj	Dup	Zero
Adv	0	0	80	68	76	0	0	0	0	0	0	0	0	0
Punc	300	0	77	71	81	1	0	0	0	0	0	0	0	0
Noun	0	10	660	472	545	11	0	0	0	0	0	0	0	0
Adj	0	2	153	165	136	1	0	0	0	0	0	0	0	0
Verb	0	42	383	248	394	0	0	0	0	0	0	0	0	0
Det	0	0	60	37	106	0	0	0	0	0	0	0	0	0
Pron	0	0	29	24	23	2	0	0	0	0	0	0	0	0
Postp	0	0	36	24	19	0	0	0	0	0	0	0	0	0
Conj	0	0	53	32	47	0	0	0	0	0	0	0	0	0
Ques	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Num	0	0	20	9	13	0	0	0	0	0	0	0	0	0
Interj	0	0	0	0	2	0	0	0	0	0	0	0	0	0
Dup	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Zero	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TABLE III: Confusion matrix for CPOSTAG type. Rows represent true tags, columns represent estimated tags.

VI. COMMENTS ON THE RESULTS

There are two remarkable observations in this experiment. First of all, the overall performance is particularly weak for such a task. When I run the same algorithm with another implementation, it is clearly seen that performance may reach to 60%. This points to a bug or a mis-implementation in the code.

Secondly, confusion matrices show never used tags in the validation set. When manually checked, there is no never-used-tags in the validation set. Again, this points to a mis-implementation.

VII. NOTES AND FUTURE WORK

It is worth to state that instead of using a gold standard file, I put true tags into evaluation script input file for convenience. Another mark is that for unknown words, I take probability as 0.0, which leads to broken Markov series.

There are missing points and obvious bugs in the code, but I left them with concerning the tight time schedule.

As for the future work, first step would be the fixing obvious bugs. Then the next step is testing the tagger with a larger validation set. A cross validation with another tagger would be a nice playground.

VIII. CONCLUSION

To conclude, in this study, I learned how to implement an HMM POS Tagger with Viterbi decoding. I evaluated the results using overall accuracy and a confusion matrix. This was a informative study.