POS tagging and smoothing

1 Introduction

The theory of Markov chains was introduced 80 years ago, and is often used in mathematics and engineering [1]. In last decade, it has been applied in speech processing, for example part-of-speech (POS) tagging. This practice is about training a hidden Markov model (HMM) and predicting the tags of the tested sentences.

To realise the target, I designed a programme by using of Python2. In the next, I explain the main parts of completing the system through two sections, programme design and accuracy assessment. In the section of programme design, the ideas of training HMM, generating POS tagger and test HMM are given, of which contains Laplace smoothing, Stupid backoff and Viterbi algorithms used in the programme. In the accuracy assessment, 6 corpora are tested in the HMM, and the accuracy results (including POS accuracy) and the evaluation of the system are shown in this part. **The section of "Instruction of running the programme " is recommended to read at first.**

2 Programme Design

2.1 The design of training HMM

All training sentence is from the first 10000 sentences of brown corpus. To achieve training the HMM, it is necessary to use a few dictionaries to save the results after training. Note that the following titles are the names of dictionaries.

(1) Pos

(codes: the 8th-22th lines)

The pos dictionary stores all POS and the number of the each POS appearing from training sentences, which is achieved by the functions of FreDist and most_common.

(2) transitions and tranProba

(codes: the 29th-42nd lines)

The transitions dictionary stores all POS bi-grams and the number of each pair occurring from training sentences. The tranProba dictionary is a collection that saves the probability of the occurrence of one POS given the prior POS, i.e. $P(t_i \mid t_{i-1})$. It uses Laplace smoothing to calculate the probabilities to ensure probabilities never become 0 even if there is a bi-grams that exists in a test sentence but not exists in the transitions dictionary. Reccomend

(3) emissions and emiProba

(codes: the 44th-80th lines)

The emissions dictionary stores all words and the corresponding POSs (each pair with these two elements is key of the dictionary), and every key is followed by the times of the occurrence of each pair. The emiProba dictionary is composed of the probability of the occurrence of one word given a POS, i.e. P(w|t). In test sentences, there might be some words that did not occur in training sentences, so this problem should be solved. The way to deal with it is that before filling up emissions dictionary, the original training sentences are replaced by the new training sentences. The words that occur only one time in original training sentences are substituted by the word named as "unknown". This new training sentences are used in the follow-up process of filling up emissions.

(4) Dictionary

(codes: the 82nd-91st lines)

In dictionary, it store all words from the new training sentences that I just talked about. These words are the keys of the dictionary, and the corresponding tags are the values of keys.

2.2 The design of POS tagger

(codes: the 93rd-145nd lines)

POS tagger is a HMM model that can tag words in sentences. Viterbi algorithms is applied in this model to decide a POS of a word. The tagger is implemented in the function named "tagger". This function can tag each sentence per

time. When starting tagging words, check if the word is in the first position of a sentence at first. If it is in the first position, P(tag | </s>) is chosen to be the transition probability , otherwise choose $P(t_i \mid t_{i-1})$. For each word in a sentence, calculate the value of the transition probability of a POS timing the emission probability of the word, and set the tag that has maximum value as the POS of the word.

2.3 The design of testing HMM

In this step, two kinds of accuracy rate are designed. Note that the following titles are the names of functions used to achieve calculating the two types of accuracy rate.

(1) CompareAccuracy

(codes: the 147th-159th lines)

In the function of CompareAccuracy, the rate of all words being tagged right is given. Through comparing real tags(real_tags) and predicted tags(tag_list), calculate the numbers of errors divided by the number of words.

(2) AccuracyOfPOS

(codes: the 161th-182th lines)

In the function of AccuracyOfPOS, the accuracy rate of each POS is calculated. Firstly, collect all POS and store them in the dictionary named compare_tags, of which each POS is a key. Count the actual number of a POS and the times of the POS that being predicted correctly, These two values are saved in a list, following the POS (i.e. the key) in the dictionary, and finally calculate the accuracy of each POS.

3 Accuracy Assessment

3.1 Accuracy of sentence

sTo measure the performance of the HMM model, Accuracies rate of tagging sentences and each POS are given. I test 6 different kinds of corpora: brown, conll2000, alpino, treebank, dependency-treebank and floresta. The records of them are shown as follows:

Hints: The trainingSeq and testSeq indicates which sentences are used to train HMM and test HMM respectively. The numOfTags and numTypePOS mean that the total number of tags of the test sentences and the number of types of POS respectively. The canUseUniversal indicates if the corpus use " tagset='universal' "

corpusName	numOfSentences	trainingSeq	testSeq	numOfTags	numTypePOS	language	canUseUniveral	accuracy
brown	57340	1-10000	10001-10051	1573	12	English	TRUE	0.918626827718
brown	57340	1-10000	10001-10501	11525	12	English	TRUE	0.907071583514
brown	57340	1-10000	10001-10051	1573	283	English	FALSE	0.797838525111
coll2000	10948	1-10000	10001-10051	1124	12	English	TRUE	0.931494661922
coll2000	10948	1-10000	10001-10501	11082	12	English	TRUE	0.924334858499
coll2000	10948	1-10000	10001-10051	1124	44	English	FALSE	0.89590747331
treebank	3914	1-3000	3001-3051	1108	12	English	TRUE	0.878158844765
treebank	3914	1-3000	3001-3501	12827	12	English	TRUE	0.826714801444
treebank	3914	1-3000	3001-3051	1108	46	English	FALSE	0.826714801444
dependency_treebank	3914	1-3000	3001-3051	1049	45	English	FALSE	0.817921830315
dependency_treebank	3914	1-3000	3001-3501	12024	45	English	FALSE	0.851463739188
floresta	9266	1-9000	9001-9051	929	269	Portuguese	FALSE	0.734122712594
alpino	7136	1-7000	7001-7051	909	17	Dutch	FALSE	0.831683168317

Table 1

From the table 1 , the conclusions is drawn:

In general,

(1) the more training sentences, the higher the accuracy

(2) the lesser types of POS, the higher the accuracy

(3) If the number of training sentences are not too many, the more test sentences, the higher the accuracy; If the number of training sentences are numerous, the more test sentences, the lower the accuracy

(4) The accuracy of English using this model performs better than Portuguese

3.2 Accuracy of each POS

Brown corpus is used to test the accuracy of each POS. The table2,3 presents the accuracy of each POS in each test. The Figure1 shows the differences of probabilities of different tags clearly, namely, the tag, ' . ' , has the highest accuracy reaching to 1, and ' X ' has the lowest accuracy. There are a few different types of marks, such as, " . ", "? " and " ; ", and they are used commonly and generally as punctuation marks, therefore the accuracy of ' . ' is highest. The POS, ' X ' seldom occurs in a sentence, in other word, the probability of ' X ' given prior POSs are very low, which leads to ' X ' is ignored sometimes.

	ADV	NOUN	NUM	ADP	PRON
1	0.819444444444444	0.930555555555556	0.979591836734694	0.930348258706468	0.96
2	0.666666666666667	0.966417910447761	0.8125	0.890710382513661	0.904761904761905
3	0.764705882352941	0.955465587044534	1	0.904494382022472	0.953488372093023
4	0.703703703703704	0.965822524191888	0.879284649776453	0.922621251388375	0.869158878504673
5	0.82051282051282	0.977011494252874	1	0.9	1
average probability	0.755006703536115	0.959054614298523	0.939642324888226	0.912043568657744	0.93748183107192

Table 2

ADJ	DET	.	PRT	VERB	X	CONJ
0.671232876712329	0.995670995670996	1	1	0.867724867724868	not_exist	1
0.672413793103448	0.9861111111111111	1	1	0.839622641509434	0	1
0.732142857142857	0.993548387096774	1	0.864864864864865	0.891089108910891	not_exist	1
0.533178114086147	0.985436893203884	1	0.942238267148014	0.798989113530326	0.777777777777778	0.997167138810198
0.757142857142857	0.989010989010989	1	0.888888888888889	0.752293577981651	1	0.964285714285714
0.673222099637528	0.989955675218751	1	0.939198404180354	0.829943861931434	0.592592592592593	0.992290570619182

Table 3

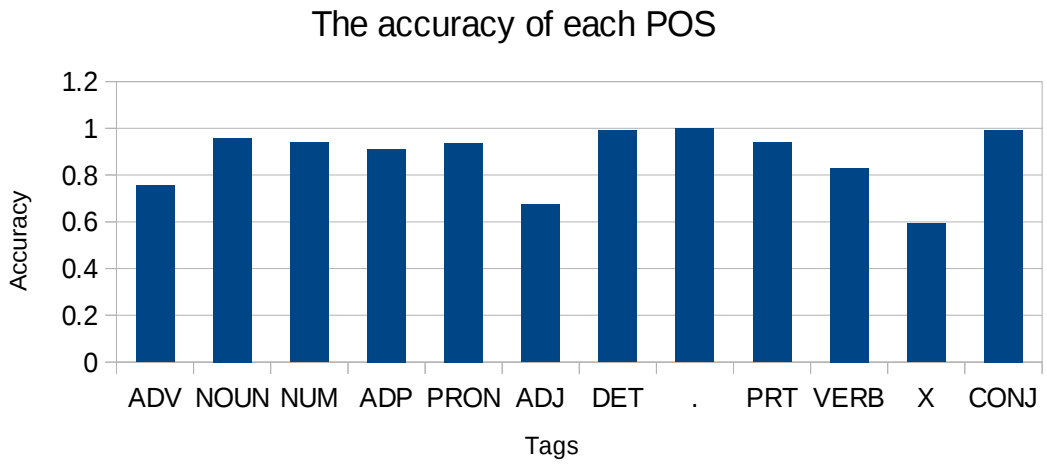

Figure 1

4 Instruction of running the programme

(1) All basic requirements and one enhancement of stupid backoff has been completed.

(2) If you want to change the training corpus, you could replace the old corpus name to a new corpus name in the 9th line, and you can choose test sentences by changing the numbers in the 186th line

(3) The submitted programme uses stupid backoff. If you want to use Laplace smoothing, just recover the 41st line, delete 42nd line, recover the 115th line, delete 116th line, recover the 136th line, delete 137th line

(4) It is recommended that it is easier and more clear to read this report with the programme codes.

5 Reference

1. Rabiner L, Juang B. An introduction to hidden Markov models. ieee assp magazine. 1986 Jan;3(1):4-16.