POS Tagging and Hidden Markov Model

h2kinfosys.com/blog/pos-tagging-and-hidden-markov-model

September 17, 2020

Artificial Intelligence Tutorials

From elementary school, we were taught to identify the parts of speech in a sentence. The fancy word for this process is called POS tagging. POS tagging may seem like an easy task but in reality, it can get really messy. In some situations, one word may have different meanings and by extension, different parts of speech, based on its context. Take a look at the sentences below

1. The man wore a bow tie to the meeting.
2. A bow and arrow were all the hunter needed to bring down the bear.
3. Villagers are expected to bow at the sight of a royal one.

In these three sentences, the word 'bow' has three different meanings and in fact, three different parts of speech. These diverging meanings make it difficult to generically assign POS tags to words in a sentence. Another reason for the difficulty faced in generic POS tagging is the birthing new words over time. For example, the word Google was not recognized as an English word in the dictionary some 20 years back. Now, 'google' is a verb that indicates the search for information on the internet using the Google search engine. This is why it is practically not possible to have a generic mapping for part of speech tags – scaling is laborious.

To overcome this, machine learning models offer a more convenient way to POS tagging. The NLTK library provides an easy way to carry out POS tagging. Let's see an example.

"Some drivers desert their passengers when they get to the desert"

In the statement above, the word 'desert' had two different POS based on its context. Will NLTK spot this variance? Let's find out.

```
#import the nltk library
import nltk
#define the text
text = 'Some drivers desert their passengers when they get to the desert'
#define an empty list to store the words and tags
tags = []
#tokenize the sentence
tokenized_sentence = nltk.sent_tokenize(text)
#loop over the list of sentence
for tokenized_text in tokenized_sentence:
    #tokenize the sentence into list of words
    tokens = nltk.word_tokenize(tokenized_text)
    #POS tag the words
    tokens = nltk.pos_tag(tokens)
    #Populate each word and its corresponding tag to the list
    tags.append(tokens)
#print the list
print(tags)
```

Output:

```
[[('Some', 'DT'), ('drivers', 'NNS'), ('desert', 'VBP'), ('their', 'PRP$'),
('passengers', 'NNS'), ('when', 'WRB'), ('they', 'PRP'), ('get', 'VBP'), ('to',
'TO'), ('the', 'DT'), ('desert', 'NN')]]
```

Sure, it did. The first 'desert' was seen as a present tense verb while the second, as a singular noun.

POS tagging is a required step in virtually all NLP applications. When building models that do speech recognition, bot chatting, language translation, and other NLP related models, you must first POS tagging the words in your corpus to have a feel of their role in the sentence.

With this understanding of what POS tagging is and why it is critical, let's delve a little deeper to understand how assigning POS tags to the words is done.

Types of POS Tagging Algorithms

In the corpus, there are two distinctive types of POS tagging algorithms (or POS taggers for short).

- Rule-based POS taggers
- Stochastic POS Tagging

Rule-Based POS Tagging

In a rule-based POS tagging approach, contextual information is used to assign POS tags to ambiguous words. When a word is unknown, the algorithm analyses the linguistic feature of the word. For a given word, it takes into cognizance the preceding words, the subsequent

words, and their POS tags. With some predetermined rules, it assigns the POS to the unknown words. This is why it is called rule-based taggers.

Determining the set of rules can be a herculean task, with serious scaling bottleneck. It then becomes imperative to find automatic ways to define these rules. In a paper published by E. Brill in 1992, he overcame the limitations of rule-based taggers and came up with a robust and automatic way of defining the rules for POS tagging. Rules such as: If the ambiguous word is preceded by a determiner, and the word following is a noun, then the ambiguous word is an adjective.

Brill's taggers trained a large dataset and found out the rules to use per time, with minimal tagging errors. The algorithm however needs a large dataset with the rule templates to automatically create new rules for new features.

Stochastic POS Tagging

The word 'stochastic' basically means anything that involves probability or statistics. Can you guess what stochastic POS is? You guessed right! Stochastic POS taggers involve approaches that incorporate frequency or probability for POS tagging. In stochastic POS tagging algorithms, after a model is trained based on some tagged datasets, the test sentence is tagged based on the frequency with which a word is tagged in the trained dataset. This may be great for individual words but can provide wrong results for a set of tags in a corpus.

The problem of not taking context into cognizance can be solved using the probability-based stochastic tagger, sometimes called the n-gram approach. In this case, the algorithm assigns tags based on the probability that a tag was assigned in the trained dataset. This approach produces better results than the former since it tags each word based on its context.

To get an even better result, the two approaches (frequency-based and probability-based measurement) can be combined. This is called Hidden Markov Models. Let's understand what Markov Models are and go further to discuss Hidden Markov Models or HMMs

Markov Model

A Markov model is a model that assumes that the possibility of a future state depends on the current state alone and not past events. This assumption is called the Markov property. So in other words, a Markov model obeys the Markov Property.

Now, let's understand what Hidden Markov Models are.

POS Tagging with Hidden Markov Model

Hidden Markov Model (HMM) is a popular stochastic method for Part of Speech tagging. HMMs are used in reinforcement learning and have wide applications in cryptography, text recognition, speech recognition, bioinformatics, and many more.

Say, for instance, we have a sentence “I love Artificial Intelligence” and we need to assign POS tags to each word. It is clear that the POS tags for each word are “Pronoun (PRP), Verb (VBP), Adjective (JJ), Noun (NN) respectively. To calculate the probabilities associated with the tags, we need to first know how likely it is for a pronoun to be followed by a verb, then an adjective, and finally a noun. These probabilities are typically called **transitions probabilities**. Secondly, we need to know how likely that the word ‘’I” would be a pronoun, the word “love” would be a verb, the word “Artificial” would be an adjective, and the word “Intelligence” would be a noun. These probabilities are called **emission probabilities**. So, what are transition and emission probabilities?

- The transition probability is the probability that connects the change from one state to the next in the system.
- The emission probability is the probability that quantifies the possibility of making a particular observation given a defined state.

The probabilities can be depicted using matrices. A transition matrix is a matrix in which the elements represent that rates in the transition from one state to another. The elements of the emission matrix, on the other hand, shows the rate of emission from one state to another.

Say we are dealing with this corpus:

“Python is easier. Artificial Intelligence with Python is great. I love Artificial Intelligence. So, I learn python”.

Counting the number of times each word is assigned to a POS tag in the corpus, the emission matrix would be:

	NN	VB	DT	JJ	PRP
Python	3	0	0	0	0
is	0	2	0	0	0
easier	0	0	0	1	0
Artificial	0	0	0	2	0
Intelligence	2	0	0	0	0
great	0	0	0	1	0
I	0	0	0	0	2
love	0	1	0	0	0
learn	0	1	0	0	0

To get the emission probabilities, each column is divided by the sum of entries in that column. Doing that, the emission probability would be:

	NN	VB	DT	JJ	PRP
Python	3/5	0	0	0	0
is	0	1/2	0	0	0
easy	0	0	0	¼	0
Artificial	0	0	0	1/2	0
Intelligence	2/5	0	0	0	0
great	0	0	0	1/4	0
I	0	0	0	0	1
love	0	1/4	0	0	0
learn	0	1/4	0	0	0

What does this translate to? It means that if we know that a word is a noun, the probability that it would be “Python” is 3/5, and the probability that it would be “Intelligence” is 2/5. This same process applies to other parts of speech.

To determine the transition probabilities, we quantify the possibility that a part of speech is followed by another part of speech. In our earlier example, the table would indicate the probability that a noun will be followed by a verb, a verb will be followed by a determiner, a

determiner will be followed by a preposition, and so on.

Having found these, the hidden Markov models can be built by mapping the words (called observations) to the part of speech (called hidden state) using the emission probabilities. Furthermore, the transition probabilities are also used to link the hidden states together.

You'd realize that when building HMM, the statistical requirements are cumbersome. This example above was for 4 short sentences. Imagine we had a book with thousands of words! The counting process would be manually impossible. NLTK has methods that help in counting the number of tags and performing other statistical tasks. We shall discuss that in the next tutorial.

Facebook Comments